

Distributed Cache Service

User Guide

Date 2022-04-12

Contents

1 Service Overview	1
1.1 What Is DCS?	1
1.2 Application Scenarios	3
1.3 DCS Instance Types	5
1.3.1 Single-Node Redis	5
1.3.2 Master/Standby Redis	7
1.3.3 Proxy Cluster Redis	9
1.3.4 Redis Cluster	11
1.3.5 Single-Node Memcached	13
1.3.6 Master/Standby Memcached	15
1.4 DCS Instance Specifications	16
1.4.1 Redis 3.0 Instance Specifications	17
1.4.2 Redis 4.0 and 5.0 Instance Specifications	18
1.4.3 Memcached Instance Specifications	26
1.5 Command Compatibility	28
1.5.1 Redis 3.0 Commands	28
1.5.2 Redis 4.0 Commands	32
1.5.3 Redis 5.0 Commands	35
1.5.4 Web CLI Commands	39
1.5.5 Memcached Commands	43
1.5.6 Command Restrictions for Cluster Instances	48
1.5.7 Other Command Usage Restrictions	49
1.6 HA and DR Policies	50
1.7 Comparing Redis Versions	53
1.8 Comparing Redis and Memcached	54
1.9 Comparing DCS and Open-Source Cache Services	55
1.10 Basic Concepts	59
1.11 Permissions Management	60
1.12 Related Services	63
2 Permissions Management	66
2.1 Creating a User and Granting DCS Permissions	66
2.2 DCS Custom Policies	67
3 Getting Started	69

3.1 Creating an Instance.....	69
3.1.1 Identifying Requirements.....	69
3.1.2 Preparing the Environment.....	70
3.1.3 Creating a DCS Redis Instance.....	71
3.1.4 Creating a DCS Memcached Instance.....	73
3.2 Accessing an Instance.....	75
3.2.1 Accessing a DCS Redis Instance Through redis-cli.....	75
3.2.2 Access in Different Languages.....	77
3.2.2.1 Java.....	78
3.2.2.1.1 Jedis.....	78
3.2.2.1.2 Lettuce.....	81
3.2.2.1.3 Redisson.....	82
3.2.2.2 Lettuce Integration with Spring Boot.....	86
3.2.2.3 Clients in Python.....	91
3.2.2.4 go-redis.....	94
3.2.2.5 hiredis in C++.....	95
3.2.2.6 C#.....	98
3.2.2.7 PHP.....	99
3.2.2.7.1 phpredis.....	100
3.2.2.7.2 Predis.....	102
3.2.2.8 Node.js.....	103
3.2.3 Accessing a DCS Redis 4.0 or 5.0 Instance on the Console.....	106
3.2.4 Accessing a DCS Memcached Instance.....	107
3.3 Viewing Details of a DCS Instance.....	108
4 Operation Guide.....	111
4.1 Operating DCS Instances.....	111
4.1.1 Modifying DCS Instance Specifications.....	111
4.1.2 Restarting DCS Instances.....	114
4.1.3 Deleting DCS Instances.....	115
4.1.4 Performing a Master/Standby Switchover for a DCS Instance.....	116
4.1.5 Clearing DCS Instance Data.....	117
4.1.6 Exporting DCS Instance List.....	117
4.1.7 Command Renaming.....	118
4.2 Managing DCS Instances.....	118
4.2.1 Configuration Notice.....	118
4.2.2 Modifying Configuration Parameters.....	119
4.2.3 Modifying Maintenance Time Window.....	129
4.2.4 Modifying the Security Group.....	129
4.2.5 Viewing Background Tasks.....	130
4.2.6 Viewing Data Storage Statistics of a DCS Redis 3.0 Proxy Cluster Instance.....	131
4.2.7 Managing Tags.....	131
4.2.8 Managing Shards and Replicas.....	133

4.2.9 Cache Analysis.....	133
4.2.10 Managing IP Address Whitelist.....	136
4.2.11 Viewing Redis Slow Queries.....	137
4.2.12 Viewing Redis Run Logs.....	138
4.2.13 Diagnosing an Instance.....	138
4.3 Backing Up and Restoring DCS Instances.....	139
4.3.1 Overview.....	139
4.3.2 Configuring a Backup Policy.....	141
4.3.3 Manually Backing Up a DCS Instance.....	142
4.3.4 Restoring a DCS Instance.....	143
4.3.5 Downloading a Backup File.....	144
4.4 Migrating Data with DCS.....	145
4.4.1 Introduction to Migration with DCS.....	145
4.4.2 Importing Backup Files.....	147
4.4.2.1 Importing Backup Files from an OBS Bucket.....	147
4.4.2.2 Importing Backup Files from Redis.....	150
4.4.3 Migrating Data Online.....	151
4.5 Managing Passwords.....	154
4.5.1 DCS Instance Passwords.....	154
4.5.2 Changing Instance Passwords.....	155
4.5.3 Resetting Instance Passwords.....	156
4.5.4 Changing Password Settings for DCS Redis Instances.....	157
4.5.5 Changing Password Settings for DCS Memcached Instances.....	158
5 Monitoring.....	159
5.1 DCS Metrics.....	159
5.2 Viewing DCS Monitoring Metrics.....	210
5.3 Configuring Alarm Rules for Critical Metrics.....	211
6 Auditing.....	214
6.1 Operations That Can Be Recorded by CTS.....	214
6.2 Viewing Traces on the CTS Console.....	217
7 FAQs.....	219
7.1 Instance Types/Versions.....	219
7.1.1 Comparing Versions.....	219
7.1.2 New Features of DCS for Redis 4.0.....	220
7.1.3 New Features of DCS for Redis 5.0.....	224
7.2 Client and Network Connection.....	231
7.2.1 Security Group Configurations.....	231
7.2.2 Does DCS Support Public Access?.....	232
7.2.3 Does DCS Support Cross-VPC Access?.....	232
7.2.4 What Should I Do If Access to DCS Fails After Server Disconnects?.....	232
7.2.5 Why Do Requests Sometimes Time Out in Clients?.....	233

7.2.6 What Should I Do If an Error Is Returned When I Use the Jedis Connection Pool?.....	233
7.2.7 Why Is "ERR unknown command" Displayed When I Access a DCS Redis Instance Through a Redis Client?.....	234
7.2.8 How Do I Access a DCS Redis Instance Through Redis Desktop Manager?.....	235
7.2.9 What If "ERR Unsupported CONFIG subcommand" is Displayed in SpringCloud?.....	236
7.2.10 How Do I Troubleshoot Redis Connection Failures?.....	237
7.2.11 What Should Be Noted When Using Redis for Pub/Sub?.....	238
7.3 Redis Usage.....	238
7.3.1 Why Is CPU Usage of a DCS Redis Instance 100%?.....	238
7.3.2 Can I Change the VPC and Subnet for a DCS Redis Instance?.....	238
7.3.3 Why Aren't Security Groups Configured for DCS Redis 4.0 and 5.0 Instances?.....	239
7.3.4 Do DCS Redis Instances Limit the Size of a Key or Value?.....	239
7.3.5 Can I Obtain the Addresses of the Nodes in a Cluster DCS Redis Instance?.....	239
7.3.6 Why Is Available Memory of a DCS Redis 3.0 Instance Smaller Than Instance Cache Size?.....	239
7.3.7 Does DCS for Redis Support Multiple Databases?.....	239
7.3.8 Does DCS for Redis Support Redis Clusters?.....	240
7.3.9 Does DCS for Redis Support Sentinel?.....	240
7.3.10 What Is the Default Data Eviction Policy?.....	240
7.3.11 What Should I Do If an Error Occurs in Redis Exporter?.....	241
7.3.12 Why Is Memory Usage More Than 100%?.....	241
7.3.13 Why Is Redisson Distributed Lock Not Supported by DCS Proxy Cluster Redis 3.0 Instances?.....	241
7.3.14 Can I Customize or Change the Port for Accessing a DCS Instance?.....	241
7.3.15 Can I Modify the Connection Addresses for Accessing a DCS Instance?.....	242
7.3.16 Does DCS Support Cross-AZ Deployment?.....	242
7.3.17 Why Does It Take a Long Time to Start a Cluster DCS Instance?.....	242
7.3.18 Does DCS for Redis Provide Backend Management Software?.....	242
7.3.19 Why Is Memory of a DCS Redis Instance Used Up by Just a Few Keys?.....	242
7.3.20 Can I Recover Data from Deleted DCS Instances?.....	242
7.3.21 Why Is "Error in execution" Returned When I Access Redis?.....	243
7.4 Redis Commands.....	243
7.4.1 How Do I Clear Redis Data?.....	243
7.4.2 How Do I Rename High-Risk Commands?.....	244
7.4.3 Does DCS for Redis Support Pipelining?.....	244
7.4.4 Does DCS for Redis Support the INCR and EXPIRE Commands?.....	244
7.4.5 Why Do I Fail to Execute Some Redis Commands?.....	244
7.4.6 Why Does a Redis Command Fail to Take Effect?.....	244
7.4.7 Is There a Time Limit on Executing Redis Commands? What Will Happen If a Command Times Out?.....	245
7.5 Instance Scaling and Upgrade.....	245
7.5.1 Can DCS Redis Instances Be Upgraded, for Example, from Redis 3.0 to Redis 4.0 or 5.0?.....	245
7.5.2 Are Services Interrupted If Maintenance is Performed During the Maintenance Time Window?.....	245
7.5.3 Are Instance Resources Affected During Specification Modification?.....	245
7.5.4 Are Services Interrupted During Specification Modification?.....	246

7.5.5 Why Can't I Modify Specifications for a DCS Redis/Memcached Instance?.....	246
7.6 Monitoring and Alarm.....	246
7.6.1 Does Redis Support Command Audits?.....	247
7.6.2 What Should I Do If the Monitoring Data of a DCS Redis Instance Is Abnormal?.....	247
7.6.3 Why Is Available Memory of Unused DCS Instances Less Than Total Memory and Why Is Memory Usage of Unused DCS Instances Greater Than Zero?.....	247
7.6.4 Why Is Used Memory Greater Than Available Memory?.....	247
7.7 Data Backup, Export, and Migration.....	247
7.7.1 How Do I Export DCS Redis Instance Data?.....	247
7.7.2 Can I Export Backup Data of DCS Redis Instances to RDB Files Using the Console?.....	248
7.7.3 Does DCS Support Data Persistence?.....	248
7.7.4 Online Migration with Rump.....	248
7.8 Master/Standby Switchover.....	250
7.8.1 When Does a Master/Standby Switchover Occur?.....	250
7.8.2 How Does Master/Standby Switchover Affect Services?.....	250
7.8.3 Does the Client Need to Switch the Connection Address After a Master/Standby Switchover?.....	250
7.8.4 How Does Redis Master/Standby Replication Work?.....	250
7.9 Memcached Usage.....	251
7.9.1 Can I Dump DCS Memcached Instance Data for Analysis?.....	251
7.9.2 What Memcached Version Is Compatible with DCS for Memcached?.....	251
7.9.3 What Data Structures Does DCS for Memcached Support?.....	251
7.9.4 Does DCS for Memcached Support Public Access?.....	251
7.9.5 Can I Modify Configuration Parameters of DCS Memcached Instances?.....	251
7.9.6 What Are the Differences Between DCS for Memcached and Self-Hosted Memcached?.....	251
7.9.7 What Policies Does DCS for Memcached Use to Deal with Expired Data?.....	252
7.9.8 How Should I Select AZs When Creating a DCS Memcached Instance?.....	252
A Change History.....	254

1 Service Overview

1.1 What Is DCS?

Distributed Cache Service (DCS) is an online, distributed, in-memory cache service compatible with Redis and Memcached. It is reliable, scalable, usable out of the box, and easy to manage, meeting your requirements for high read/write performance and fast data access.

- Usability out of the box
DCS provides single-node, master/standby, and cluster instances with specifications ranging from 128 MB to 1024 GB. DCS instances can be created with just a few clicks on the console, without requiring you to prepare servers. DCS Redis 4.0 and 5.0 instances are containerized and can be created within seconds.
- Security and reliability
Instance data storage and access are securely protected through security management services, including Identity and Access Management (IAM), Virtual Private Cloud (VPC), Cloud Eye, and Cloud Trace Service (CTS). Master/Standby and cluster instances can be deployed within an availability zone (AZ) or across AZs.
- Auto scaling
DCS instances can be scaled up or down online, helping you control costs based on service requirements.
- Easy management
A web-based console is provided for you to perform various operations, such as restarting instances, modifying configuration parameters, and backing up and restoring data. RESTful application programming interfaces (APIs) are also provided for automatic instance management.
- Online migration
You can create a data migration task on the console to import backup files or migrate data online.

DCS for Redis

Redis is a storage system that supports multiple types of data structures, including key-value pairs. It can be used in such scenarios as data caching, event publication/subscription, and high-speed queuing, as described in [Application Scenarios](#). Redis is written in ANSI C, supporting direct read/write of [strings](#), [hashes](#), [lists](#), [sets](#), [sorted sets](#), and [streams](#). Redis works with an in-memory dataset which can be persisted on disk.

DCS Redis instances can be customized based on your requirements.

Table 1-1 DCS Redis instance configuration

<p>Instance type</p>	<p>DCS for Redis provides the following types of instances to suit different service scenarios:</p> <p>Single-node: Suitable for caching temporary data in low reliability scenarios. Single-node instances support highly concurrent read/write operations, but do not support data persistence. Data will be deleted after instances are restarted.</p> <p>Master/Standby: Each master/standby instance runs on two nodes (one master and one standby). The standby node replicates data synchronously from the master node. If the master node fails, the standby node automatically becomes the master node.</p> <p>Proxy Cluster: In addition to the native Redis cluster, a Proxy Cluster instance has proxies and load balancers. Load balancers implement load balancing. Different requests are distributed to different proxies to achieve high-concurrency. Each shard in the cluster has a master node and a standby node. If the master node is faulty, the standby node on the same shard is promoted to the master role to take over services.</p> <p>Redis Cluster: Each Redis Cluster instance consists of multiple shards and each shard includes a master node and multiple replicas (or no replica at all). Shards are not visible to you. If the master node fails, a replica on the same shard takes over services. You can split read and write operations by writing to the master node and reading from the replicas. This improves the overall cache read/write performance.</p>
<p>Instance specification</p>	<p>DCS for Redis provides instances of different specifications, ranging from 128 MB to 1024 GB.</p>
<p>Redis version</p>	<p>DCS instances are compatible with open-source Redis 3.0, 4.0, and 5.0.</p>
<p>Underlying architecture</p>	<p>Standard Redis based on VMs: supports up to 100,000 queries per second (QPS) at a single node.</p>

High availability (HA) and DR	Master/standby and cluster DCS Redis instances can be deployed across AZs in the same region with physically isolated power supplies and networks.
--------------------------------------	--

For more information about open-source Redis, visit <https://redis.io/>.

DCS for Memcached

Memcached is an in-memory key-value caching system that supports read/write of simple strings. It is often used to cache backend database data to alleviate load on these databases and accelerate web applications. For details about its application scenarios, see [Memcached Application Scenarios](#).

In addition to full compatibility with Memcached, DCS for Memcached provides the hot standby and data persistence.

Table 1-2 DCS Memcached instance configuration

Instance type	DCS for Memcached provides the following two types of instances to suit different service scenarios: Single-node: Suitable for caching temporary data in low reliability scenarios. Single-node instances support highly concurrent read/write operations, but do not support data persistence. Data will be deleted after instances are restarted. Master/Standby: Each master/standby instance runs on two nodes (one master and one standby). The standby node replicates data synchronously from the master node, but does not support read/write operations. If the master node fails, the standby node automatically becomes the master node.
Memory	Specification of single-node or master/standby DCS Memcached instances: 2 GB, 4 GB, 8 GB, 16 GB, 32 GB, and 64 GB.
HA and DR	Master/Standby DCS Memcached instances can be deployed across AZs in the same region with physically isolated power supplies and networks.

For more information about open-source Memcached, visit <https://memcached.org/>.

1.2 Application Scenarios

Redis Application Scenarios

Many large-scale e-commerce websites and video streaming and gaming applications require fast access to large amounts of data that has simple data structures and does not need frequent join queries. In such scenarios, you can use

Redis to achieve fast yet inexpensive access to data. Redis enables you to retrieve data from in-memory data stores instead of relying entirely on slower disk-based databases. In addition, you no longer need to perform additional management tasks. These features make Redis an important supplement to traditional disk-based databases and a basic service essential for internet applications receiving high-concurrency access.

Typical application scenarios of DCS for Redis are as follows:

1. **E-commerce flash sales**

E-commerce product catalogue, deals, and flash sales data can be cached to Redis.

For example, the high-concurrency data access in flash sales can be hardly handled by traditional relational databases. It requires the hardware to have higher configuration such as disk I/O. By contrast, Redis supports 100,000 QPS per node and allows you to implement locking using simple commands such as **SET**, **GET**, **DEL**, and **RPOUSH** to handle flash sales.

2. **Live video commenting**

In live streaming, online user, gift ranking, and bullet comment data can be stored as sorted sets in Redis.

For example, bullet comments can be returned using the **ZREVRANGEBYSCORE** command. The **ZPOPMAX** and **ZPOPMIN** commands in Redis 5.0 can further facilitate message processing.

3. **Game leaderboard**

In online gaming, the highest ranking players are displayed and updated in real time. The leaderboard ranking can be stored as sorted sets, which are easy to use with up to 20 commands.

4. **Social networking comments**

In web applications, queries of post comments often involve sorting by time in descending order. As comments pile up, sorting becomes less efficient.

By using lists in Redis, a preset number of comments can be returned from the cache, rather than from disk, easing the load off the database and accelerating application responses.

Memcached Application Scenarios

Memcached is suitable for storing simple key-value data.

1. **Web pages**

Caching static data such as HTML pages, Cascading Style Sheets (CSS), and images to DCS Memcached instances improves access performance of web pages.

2. **Frontend database**

In dynamic systems such as social networking and blogging sites, write operations are far fewer than read operations such as querying users, friends, and articles. Such frequently access data can be cached in Memcached to reduce database load and improve performance.

The following data can be cached:

- Frequently accessed data that does not require real-time updates and can expire automatically

Example: latest article lists and rankings. Although data is generated constantly, its impact on user experience is limited. Such data can be cached for a preset period of time and accessed from the database after this period. If web page editors want to view the latest ranking, a cache clearing or refreshing policy can be configured.

- Frequently accessed data that requires real-time updates

Example: friend lists, article lists, and reading records. Such data can be cached to Memcached first, and then updated whenever changes (adding, modifying, and deleting data) occur.

3. Flash sales

It is difficult for traditional databases to write an order placement operation during flash sales into the database, modify the inventory data, and ensure transaction consistency while ensuring uninterrupted user experience.

Memcached **incr** and **decr** commands can be used to store inventory information and complete order placement in memory. Once an order is submitted, an order number is generated. Then, the order can be paid.

NOTE

Scenarios where Memcached is not suitable:

- The size of a single cache object is larger than 1 MB.

Memcached cannot cache an object larger than 1 MB. In such cases, use Redis.

- The key contains more than 250 characters.

To use Memcached in such a scenario, you can generate an MD5 hash for the key and cache the hash instead.

- High data reliability is required.

Open-source Memcached does not provide data replication, backup, and migration, so data persistence is not supported.

Master/Standby DCS Memcached instances support data persistence. For more information, contact technical support.

- Complex data structures and processing are required.

Memcached supports only simple key-value pairs, and does not support complex data structures such as lists and sets, or complex operations such as sorting.

1.3 DCS Instance Types

1.3.1 Single-Node Redis

Three Redis versions are available for single-node DCS Redis instances: Redis 3.0, Redis 4.0, and Redis 5.0.

Features

1. Low system overhead and high QPS

Single-node instances do not support data synchronization or data persistence, reducing system overhead and supporting higher concurrency. QPS of single-node DCS Redis instances reaches up to 100,000.

2. Process monitoring and automatic fault recovery

With an HA monitoring mechanism, if a single-node DCS instance becomes faulty, a new process is started within 30 seconds to resume service provisioning.

3. Out-of-the-box usability and no data persistence

Single-node DCS instances can be used out of the box because they do not involve data loading. If your service requires high QPS, you can warm up the data beforehand to avoid strong concurrency impact on the backend database.

4. Low-cost and suitable for development and testing

Single-node instances are 40% cheaper than master/standby DCS instances, suitable for setting up development or testing environments.

In summary, single-node DCS instances support highly concurrent read/write operations, but do not support data persistence. Data will be deleted after instances are restarted. They are suitable for scenarios which do not require data persistence, such as database front-end caching, to accelerate access and ease the concurrency load off the backend. If the desired data does not exist in the cache, requests will go to the database. When restarting the service or the DCS instance, you can pre-generate cache data from the disk database to relieve pressure on the backend during startup.

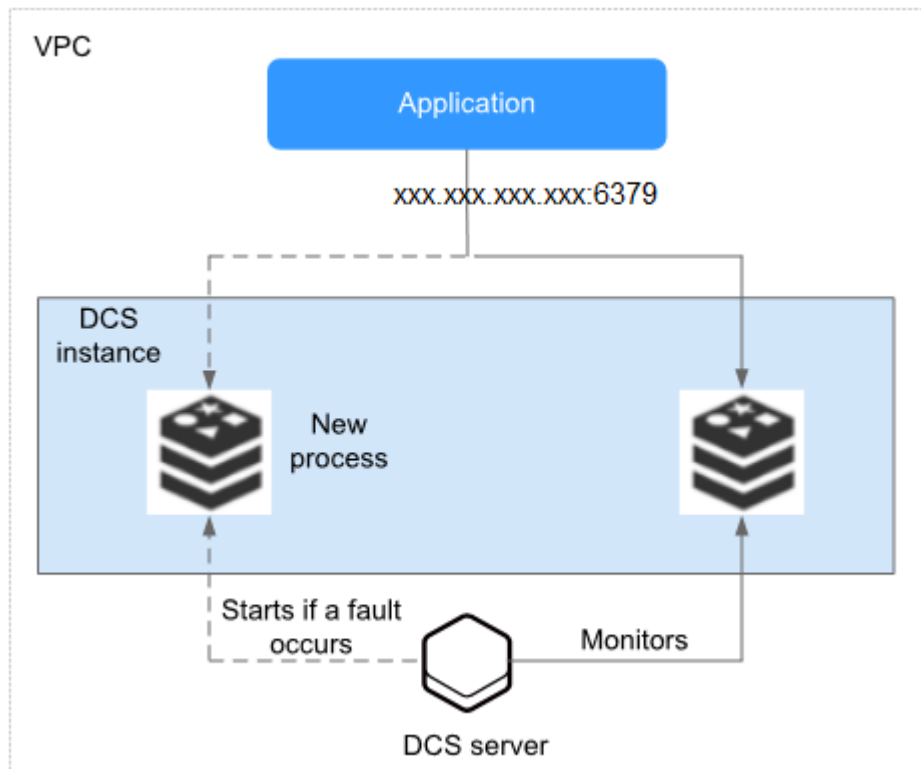
Architecture

Figure 1-1 shows the architecture of single-node DCS Redis instances.

 **NOTE**

To access a DCS Redis 3.0 instance, you must use port 6379. To access a DCS Redis 4.0 or 5.0 instance, you can customize the port. If no port is specified, the default port 6379 will be used. In the following architecture, port 6379 is used. If you have customized a port, replace **6379** with the actual port.

Figure 1-1 Single-node DCS Redis instance architecture



Architecture description:

- VPC**
 All server nodes of the instance run in the same VPC.
 - NOTE**
 For intra-VPC access, the client and the instance must be in the same VPC with specified security group rule configurations.
 For details, see [Security Group Configurations](#).
- Application**
 The client of the instance, which is the application running on an Elastic Cloud Server (ECS).
 DCS Redis instances are compatible with the Redis protocol, and can be accessed through open-source clients. For details about accessing DCS instances, see [Accessing an Instance](#).
- DCS instance**
 A single-node DCS instance, which has only one node and one Redis process.
 DCS monitors the availability of the instance in real time. If the Redis process becomes faulty, DCS starts a new process to resume service provisioning.

1.3.2 Master/Standby Redis

Both DCS for Redis and DCS for Memcached support the master/standby instance type. This section describes master/standby DCS Redis instances. Three Redis

versions are available for master/standby DCS Redis instances: Redis 3.0, Redis 4.0, and Redis 5.0.

 **NOTE**

You cannot upgrade the Redis version for an instance. For example, a master/standby DCS Redis 3.0 instance cannot be upgraded to a master/standby DCS Redis 4.0 or 5.0 instance. If your service requires the features of higher Redis versions, create a DCS Redis instance of a higher version and then migrate data from the old instance to the new one.

Features

Master/Standby DCS instances have higher availability and reliability than single-node DCS instances.

Master/Standby DCS instances have the following features:

1. **Data persistence and high reliability**

By default, data persistence is enabled by both the master and the standby node of a master/standby instance.

The standby node of a DCS Redis instance is invisible to you. Only the master node provides data read/write operations.

2. **Data synchronization**

Data in the master and standby nodes is kept consistent through incremental synchronization.

 **NOTE**

After recovering from a network exception or node fault, master/standby instances perform a full synchronization to ensure data consistency.

3. **Automatic master/standby switchover**

If the master node becomes faulty, the standby node takes over within 30 seconds, without requiring any service interruptions or manual operations.

4. **DR policies**

Each master/standby instance can be deployed across AZs with physically isolated power supplies and networks. Applications can also be deployed across AZs to achieve high availability for both data and applications.

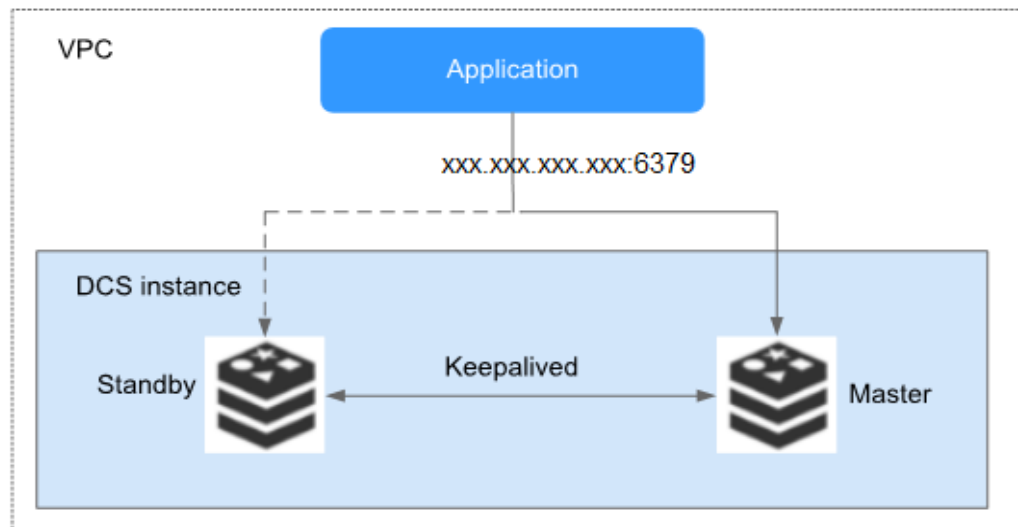
Architecture

Figure 1-2 shows the architecture of master/standby DCS Redis instances.

 **NOTE**

To access a DCS Redis 3.0 instance, you must use port 6379. To access a DCS Redis 4.0 or 5.0 instance, you can customize the port. If no port is specified, the default port 6379 will be used. In the following architecture, port 6379 is used. If you have customized a port, replace **6379** with the actual port.

Figure 1-2 Master/Standby DCS instance architecture



Architecture description:

- **VPC**

All server nodes of the instance run in the same VPC.

 **NOTE**

For intra-VPC access, the client and the instance must be in the same VPC with specified security group rule configurations.

For details, see [Security Group Configurations](#).

- **Application**

The Redis client of the instance, which is the application running on the ECS.

DCS Redis instances are compatible with the Redis protocol, and can be accessed through open-source clients. For details about accessing DCS instances, see [Accessing an Instance](#).

- **DCS instance**

Indicates a master/standby DCS instance which has a master node and a standby node. By default, data persistence is enabled and data is synchronized between the two nodes.

DCS monitors the availability of the instance in real time. If the master node becomes faulty, the standby node becomes the master node and resumes service provisioning.

DCS Redis instances are accessed through port 6379 by default.

1.3.3 Proxy Cluster Redis

DCS provides two types of cluster Redis instances: Proxy Cluster and Redis Cluster. Proxy Cluster uses Linux Virtual Server (LVS) and proxies. Redis Cluster is the native distributed implementation of Redis. Proxy Cluster instances are compatible with Redis 3.0, while Redis Cluster instances are compatible with Redis 4.0 and 5.0.

This section describes Proxy Cluster DCS Redis 3.0 instances.

NOTE

- A Proxy Cluster instance can be connected in the same way that a single-node or master/standby instance is connected, without any special settings on the client. You can use the IP address or domain name of the instance, and do not need to know or use the proxy or shard addresses.

Proxy Cluster DCS Redis 3.0 Instances

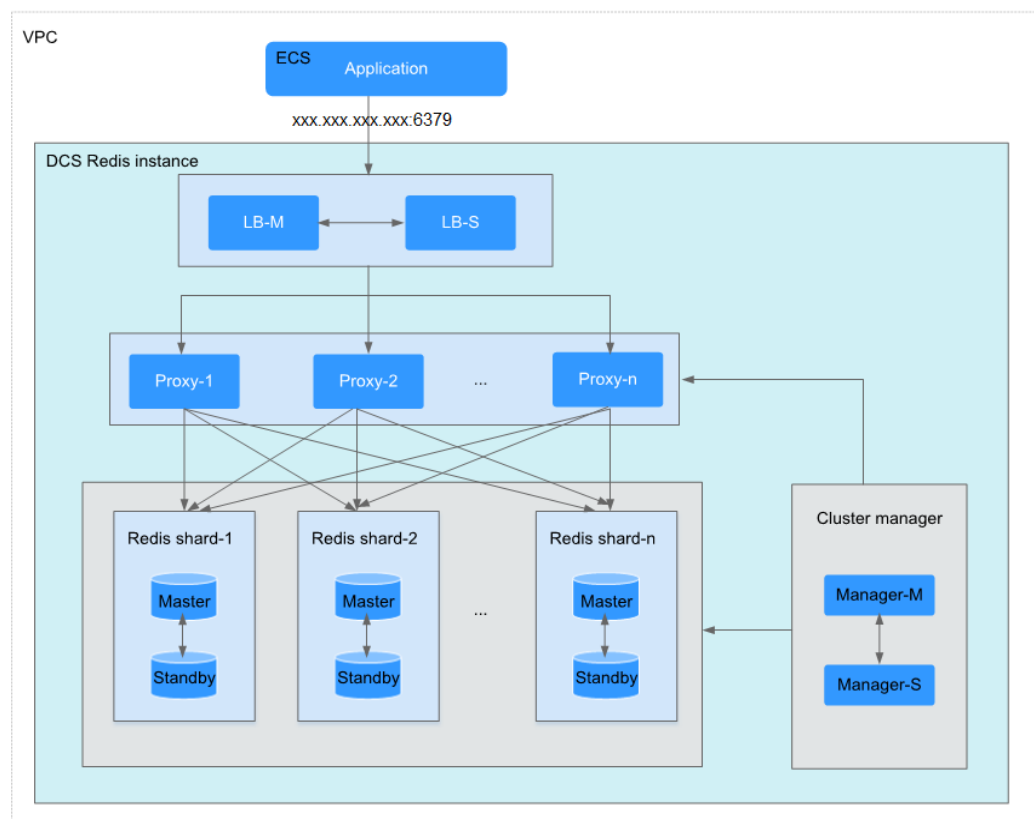
Proxy Cluster DCS Redis 3.0 instances are compatible with **codis**. The specifications range from 64 GB to 1024 GB, meeting requirements for **millions of concurrent connections** and **massive data cache**. Distributed data storage and access is implemented by DCS, without requiring development or maintenance.

Each Proxy Cluster instance consists of load balancers, proxies, cluster managers, and **shards**.

Table 1-3 Specifications of Proxy Cluster DCS Redis 3.0 instances

Total Memory	Proxies	Shards
64 GB	3	8
128 GB	6	16
256 GB	8	32

Figure 1-3 Proxy Cluster DCS Redis instance architecture



Architecture description:

- **VPC**

All server nodes of the instance run in the same VPC.

 **NOTE**

For intra-VPC access, the client and the instance must be in the same VPC with specified security group rule configurations.

For details, see [Security Group Configurations](#).

- **Application**

The client used to access the instance.

DCS Redis instances can be accessed through open-source clients. For details about accessing DCS instances, see [Accessing an Instance](#).

- **LB-M/LB-S**

The load balancers, which are deployed in master/standby HA mode. The connection addresses (**IP address:Port**) of the cluster DCS Redis instance are the addresses of the load balancers.

- **Proxy**

The proxy server used to achieve high availability and process high-concurrency client requests.

You can connect to a Proxy Cluster instance at the IP addresses of its proxies.

- **Redis shard**

A shard of the cluster.

Each shard consists of a pair of master/standby nodes. If the master node becomes faulty, the standby node automatically takes over cluster services.

If both the master and standby nodes of a shard are faulty, the cluster can still provide services but the data on the faulty shard is inaccessible.

- **Cluster manager**

The cluster configuration managers, which store configurations and partitioning policies of the cluster. You cannot modify the information about the configuration managers.

1.3.4 Redis Cluster

DCS provides two types of cluster Redis instances: Proxy Cluster and Redis Cluster. Proxy Cluster uses Linux Virtual Server (LVS) and proxies. Redis Cluster is the native distributed implementation of Redis. Proxy Cluster instances are compatible with Redis 3.0, while Redis Cluster instances are compatible with Redis 4.0 and 5.0.

This section describes Redis Cluster DCS Redis 4.0 and 5.0 instances.

Redis Cluster DCS Redis 4.0 and 5.0 Instances

The Redis Cluster instance type provided by DCS is compatible with the [native Redis Cluster](#), which uses smart clients and a distributed architecture to perform sharding.

[Table 1-4](#) lists the shard specification for different instance specifications.

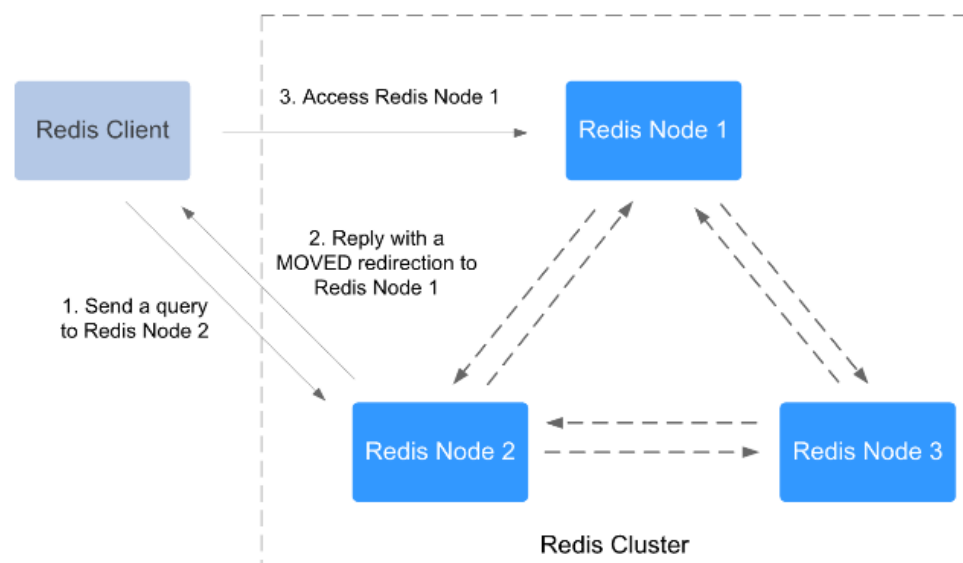
Specification per shard=Instance specification/Number of shards. For example, if a 48 GB instance has 6 shards, the specification of each shard is 48 GB/6 = 8 GB.

Table 1-4 Specifications of Redis Cluster DCS instances

Total Memory	Shards
4 GB/8 GB/16 GB/24 GB/32 GB	3
48 GB	6
64 GB	8
96 GB	12
128 GB	16
192 GB	24
256 GB	32
384 GB	48
512 GB	64
768 GB	96
1024 GB	128

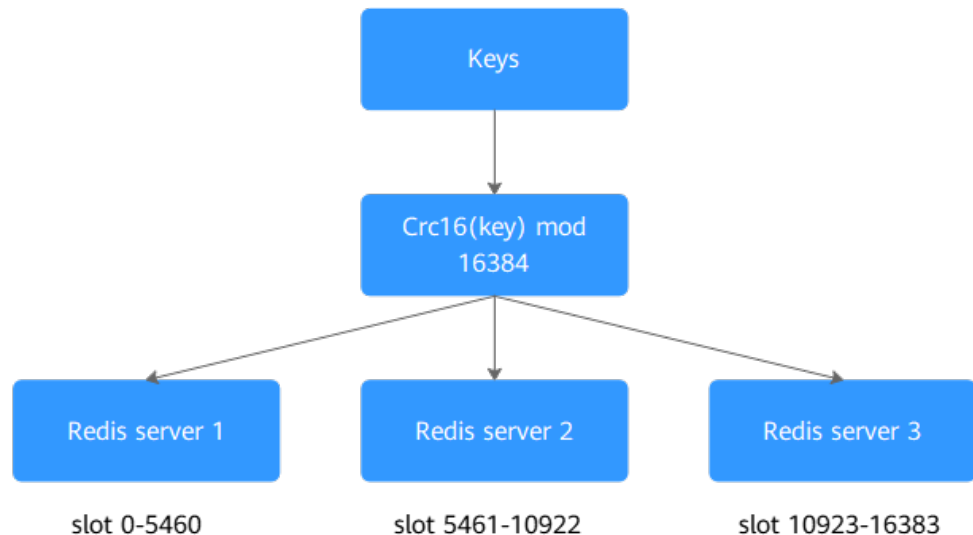
- Distributed architecture
Any node in a Redis Cluster can receive requests. Received requests are then redirected to the right node for processing. Each node consists of a subset of one master and one (by default) or multiple replicas. The master or replica roles are determined through an election algorithm.

Figure 1-4 Distributed architecture of Redis Cluster



- Presharding
There are 16,384 hash slots in each Redis Cluster. The mapping between hash slots and Redis nodes is stored in Redis Servers. To compute what is the hash slot of a given key, simply take the CRC16 of the key modulo 16384. Example command output

Figure 1-5 Redis Cluster presharding



1.3.5 Single-Node Memcached

This section describes the features and architecture of single-node DCS Memcached instances.

Features

1. Low system overhead and high QPS
Single-node instances do not support data synchronization or data persistence, reducing system overhead and supporting higher concurrency. QPS of single-node DCS Memcached instances reaches up to 100,000.
2. Process monitoring and automatic fault recovery
With an HA monitoring mechanism, if a single-node DCS instance becomes faulty, a new process is started within 30 seconds to resume service provisioning.
3. Out-of-the-box usability and no data persistence
Single-node DCS instances can be used out of the box because they do not involve data loading. If your service requires high QPS, you can warm up the data beforehand to avoid strong concurrency impact on the backend database.
4. Low-cost and suitable for development and testing
Single-node instances are 40% cheaper than master/standby DCS instances, suitable for setting up development or testing environments.

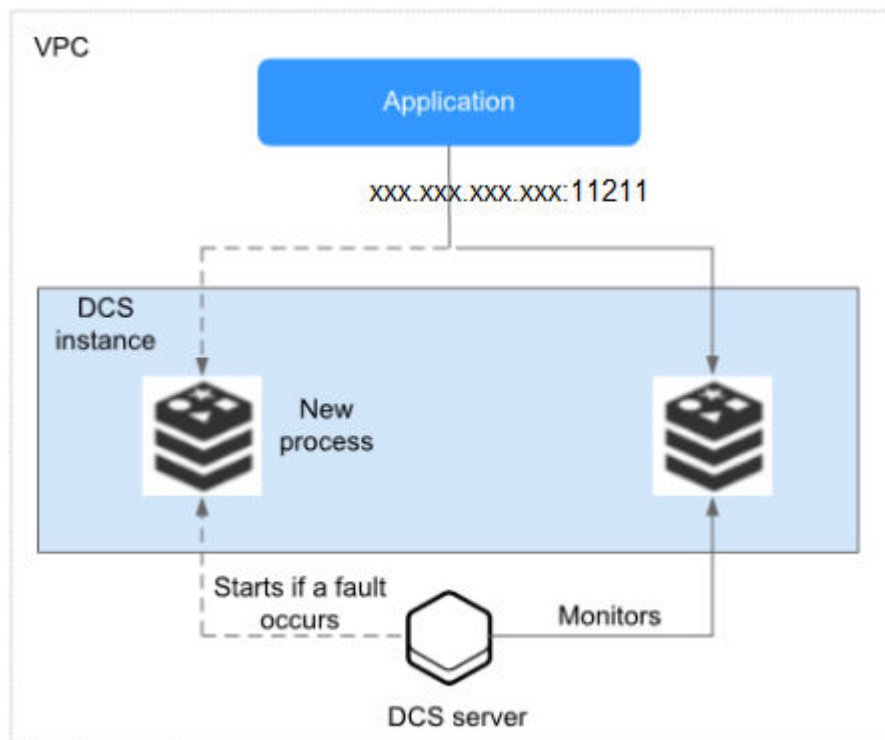
In summary, single-node DCS instances support highly concurrent read/write operations, but do not support data persistence. Data will be deleted after

instances are restarted. They are suitable for scenarios which do not require data persistence, such as database front-end caching, to accelerate access and ease the concurrency load off the backend. If the desired data does not exist in the cache, requests will go to the database. When restarting the service or the DCS instance, you can pre-generate cache data from the disk database to relieve pressure on the backend during startup.

Architecture

Figure 1-6 shows the architecture of single-node DCS Memcached instances.

Figure 1-6 Single-node DCS instance architecture



Architecture description:

- **VPC**

The VPC where all nodes of the instance are run.

NOTE

For intra-VPC access, the client and the instance must be in the same VPC with specified security group rule configurations.

For details, see [Security Group Configurations](#).

- **Application**

The client of the instance, which is the application running on an Elastic Cloud Server (ECS).

DCS Memcached instances are compatible with the Memcached protocol, and can be accessed through open-source clients. For examples of accessing DCS instances, see [Accessing a DCS Memcached Instance](#).

- **DCS instance**

A single-node DCS instance, which has only one node and one Memcached process.

DCS monitors the availability of the instance in real time. If the Memcached process becomes faulty, DCS starts a new process to resume service provisioning.

Use port 11211 to access a DCS Memcached instance.

1.3.6 Master/Standby Memcached

This section describes master/standby DCS Memcached instances.

Features

Master/Standby instances have higher availability and reliability than single-node instances.

Master/Standby DCS Memcached instances have the following features:

1. **Data persistence and high reliability**

By default, data persistence is enabled by both the master and the standby node of a master/standby DCS Memcached instance. In addition, data persistence is supported to ensure high data reliability.

The standby node of a DCS Memcached instance is invisible to you. Only the master node provides data read/write operations.

2. **Data synchronization**

Data in the master and standby nodes is kept consistent through incremental synchronization.

 **NOTE**

After recovering from a network exception or node fault, master/standby instances perform a full synchronization to ensure data consistency.

3. **Automatic master/standby switchover**

If the master node becomes faulty, the standby node takes over within 30 seconds, without requiring any service interruptions or manual operations.

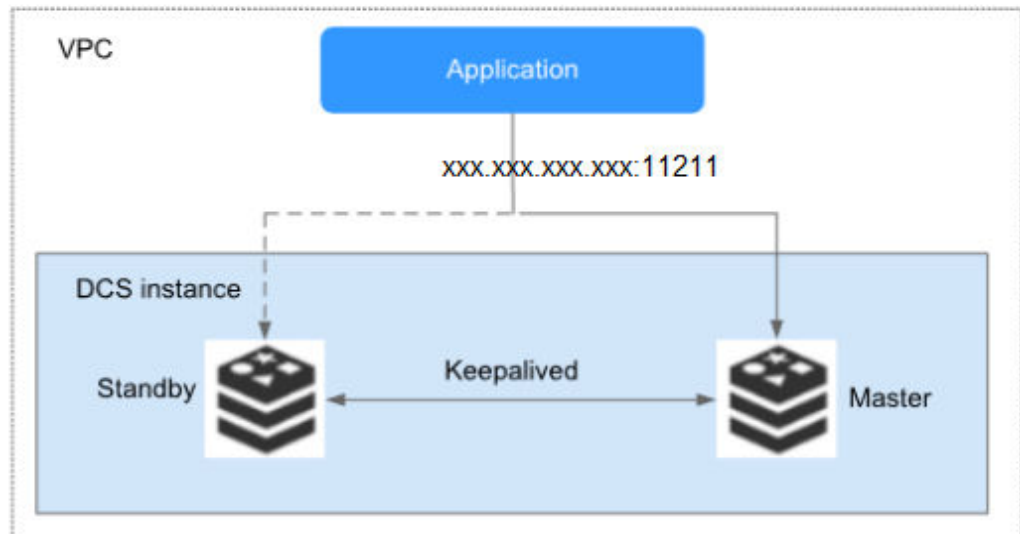
4. **Multiple DR policies**

Each master/standby instance can be deployed across AZs with physically isolated power supplies and networks. Applications can also be deployed across AZs to achieve HA for both data and applications.

Architecture of Master/Standby DCS Memcached Instances

[Figure 1-7](#) shows the architecture of master/standby DCS Memcached instances.

Figure 1-7 Master/Standby DCS Memcached instance architecture



Architecture description:

- **VPC**

The VPC where all nodes of the instance are run.

 **NOTE**

For intra-VPC access, the client and the instance must be in the same VPC with specified security group rule configurations.

For details, see [Security Group Configurations](#).

- **Application**

The Memcached client of the instance, which is the application running on the ECS.

DCS Memcached instances are compatible with the Memcached protocol, and can be accessed through open-source clients. For examples of accessing DCS instances, see [Accessing a DCS Memcached Instance](#).

- **DCS instance**

Indicates a master/standby DCS instance which has a master node and a standby node. By default, data persistence is enabled and data is synchronized between the two nodes.

DCS monitors the availability of the instance in real time. If the master node becomes faulty, the standby node becomes the master node and resumes service provisioning.

Use port 11211 to access a DCS Memcached instance.

1.4 DCS Instance Specifications

1.4.1 Redis 3.0 Instance Specifications

This section describes DCS Redis 3.0 instance specifications, including the total memory, available memory, maximum number of connections allowed, maximum/assured bandwidth, and reference performance.

The following metrics are related to the instance specifications:

- **Used memory:** You can check the memory usage of an instance by viewing the **Memory Usage** and **Used Memory** metrics.
- **Maximum connections:** The maximum number of connections allowed is the maximum number of clients that can be connected to an instance. To check the number of connections to an instance, view the **Connected Clients** metric.
- **QPS** represents queries per second, which is the number of commands processed per second.

 **NOTE**

- Single-node, master/standby, and Proxy Cluster types are available.
- Only the x86 architecture is supported. The Arm architecture is not supported.

Single-Node Instances

For each single-node DCS Redis instance, the available memory is less than the total memory because some memory is reserved for system overheads, as shown in the following table.

Table 1-5 Specifications of single-node DCS Redis 3.0 instances

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)
2	1.5	5000/50,000	42/512	50,000
4	3.2	5000/50,000	64/1536	100,000
8	6.8	5000/50,000	64/1536	100,000
16	13.6	5000/50,000	85/3072	100,000
32	27.2	5000/50,000	85/3072	100,000
64	58.2	5000/60,000	128/5120	100,000

Master/Standby Instances

For each master/standby DCS Redis instance, the available memory is less than that of a single-node DCS Redis instance because some memory is reserved for data persistence, as shown in the following table. The available memory of a

master/standby instance can be adjusted to support background tasks such as data persistence and master/standby synchronization.

Table 1-6 Specifications of master/standby DCS Redis 3.0 instances

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)
2	1.5	5000/50,000	42/512	50,000
4	3.2	5000/50,000	64/1536	100,000
8	6.4	5000/50,000	64/1536	100,000
16	12.8	5000/50,000	85/3072	100,000
32	25.6	5000/50,000	85/3072	100,000
64	51.2	5000/60,000	128/5120	100,000

Proxy Cluster Instances

In addition to larger memory, cluster instances feature more connections allowed, higher bandwidth allowed, and more QPS than single-node and master/standby instances.

Table 1-7 Specifications of Proxy Cluster DCS Redis 3.0 instances

Specificati on (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)
64	64	90,000/90,000	600/5120	500,000
128	128	180,000/180,000	600/5120	500,000
256	256	240,000/240,000	600/5120	500,000

1.4.2 Redis 4.0 and 5.0 Instance Specifications

This section describes DCS Redis 4.0 and 5.0 instance specifications, including the total memory, available memory, maximum number of connections allowed, maximum/assured bandwidth, and reference performance.

The following metrics are related to the instance specifications:

- Used memory: You can check the memory usage of an instance by viewing the **Memory Usage** and **Used Memory** metrics.
- Maximum connections: The maximum number of connections allowed is the maximum number of clients that can be connected to an instance. To check the number of connections to an instance, view the **Connected Clients** metric.
- QPS represents queries per second, which is the number of commands processed per second.
- Bandwidth: You can view the **Flow Control Times** metric to check whether the bandwidth has exceeded the limit.

 **NOTE**

- Single-node, master/standby, and Redis Cluster types are available.
- Only the x86 architecture is supported. The Arm architecture is not supported.

Single-Node Instances

Table 1-8 Specifications of single-node DCS Redis 4.0 or 5.0 instances

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
1	1	10,000/50,000	80/80	80,000	x86: redis.single.xu1.large.1 Arm: redis.single.au1.large.1
2	2	10,000/50,000	128/128	80,000	x86: redis.single.xu1.large.2 Arm: redis.single.au1.large.2

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
4	4	10,000/50,000	192/192	80,000	x86: redis.single.xu1.large.4 Arm: redis.single.au1.large.4
8	8	10,000/50,000	192/192	100,000	x86: redis.single.xu1.large.8 Arm: redis.single.au1.large.8
16	16	10,000/50,000	256/256	100,000	x86: redis.single.xu1.large.16 Arm: redis.single.au1.large.16
24	24	10,000/50,000	256/256	100,000	x86: redis.single.xu1.large.24 Arm: redis.single.au1.large.24

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
32	32	10,000/50,000	256/256	100,000	x86: redis.single.xu1.large.32 Arm: redis.single.au1.large.32
48	48	10,000/50,000	256/256	100,000	x86: redis.single.xu1.large.48 Arm: redis.single.au1.large.48
64	64	10,000/50,000	384/384	100,000	x86: redis.single.xu1.large.64 Arm: redis.single.au1.large.64

Master/Standby Instances

The following table lists the x86 and Arm specification codes (**spec_code**) when there are two default replicas. Change the replica quantity in the specification codes based on the actual number of replicas. For example, if an 8 GB master/standby x86-based instance has two replicas, its specification code is `redis.ha.xu1.large.r2.8`. If it has three replicas, its specification code is `redis.ha.xu1.large.r3.8`.

Table 1-9 Specifications of master/standby DCS Redis 4.0 or 5.0 instances

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
1	1	10,000/50,000	80/80	80,000	x86: redis.ha.xu1.large.r2.1 Arm: redis.ha.au1.large.r2.1
2	2	10,000/50,000	128/128	80,000	x86: redis.ha.xu1.large.r2.2 Arm: redis.ha.au1.large.r2.2
4	4	10,000/50,000	192/192	80,000	x86: redis.ha.xu1.large.r2.4 Arm: redis.ha.au1.large.r2.4
8	8	10,000/50,000	192/192	100,000	x86: redis.ha.xu1.large.r2.8 Arm: redis.ha.au1.large.r2.8
16	16	10,000/50,000	256/256	100,000	x86: redis.ha.xu1.large.r2.16 Arm: redis.ha.au1.large.r2.16

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
24	24	10,000/50,000	256/256	100,000	x86: redis.ha.xu1.large.r2.24 Arm: redis.ha.au1.large.r2.24
32	32	10,000/50,000	256/256	100,000	x86: redis.ha.xu1.large.r2.32 Arm: redis.ha.au1.large.r2.32
48	48	10,000/50,000	256/256	100,000	x86: redis.ha.xu1.large.r2.48 Arm: redis.ha.au1.large.r2.48
64	64	10,000/50,000	384/384	100,000	x86: redis.ha.xu1.large.r2.64 Arm: redis.ha.au1.large.r2.64

Redis Cluster Instances

In addition to larger memory, Redis Cluster instances feature more connections allowed, higher bandwidth allowed, and more QPS than single-node and master/standby instances.

The following table lists the x86 and Arm specification codes (**spec_code**) when there are two default replicas. Change the replica quantity in the specification codes based on the actual number of replicas. For example, if an 8 GB x86-based instance has two replicas, its specification code is `redis.cluster.xu1.large.r2.8`. If it has three replicas, its specification code is `redis.cluster.xu1.large.r3.8`.

Table 1-10 Specifications of Redis Cluster DCS Redis 4.0 or 5.0 instances

Total Memory (GB)	Available Memory (GB)	Shards (Master Nodes)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
4	4	3	30,000 /150,000	2304/2304	240,000	x86: redis.cluster.xu1.large.r2.4 Arm: redis.cluster.au1.large.r2.4
8	8	3	30,000 /150,000	2304/2304	240,000	x86: redis.cluster.xu1.large.r2.8 Arm: redis.cluster.au1.large.r2.8
16	16	3	30,000 /150,000	2304/2304	240,000	x86: redis.cluster.xu1.large.r2.16 Arm: redis.cluster.au1.large.r2.16

Total Memory (GB)	Available Memory (GB)	Shards (Master Nodes)	Max. Connections (Default /Limit) (Count)	Assured/ Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
32	32	3	30,000 /150,000	2304/2304	300,000	x86: redis.cluster.xu1.large.r2.32 Arm: redis.cluster.au1.large.r2.32
64	64	8	80,000 /400,000	6144/6144	500,000	x86: redis.cluster.xu1.large.r2.64 Arm: redis.cluster.au1.large.r2.64
128	128	16	160,000 /800,000	12,288/12,288	1,000,000	x86: redis.cluster.xu1.large.r2.128 Arm: redis.cluster.au1.large.r2.128

Total Memory (GB)	Available Memory (GB)	Shards (Master Nodes)	Max. Connections (Default /Limit) (Count)	Assured/ Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)	Specification Code (spec_code in the API)
256	256	32	320,000 / 1,600,000	24,576/24,576	> 2,000,000	x86: redis.cluster.xu1.large.r2.256 Arm: redis.cluster.au1.large.r2.256
512	512	64	640,000 / 3,200,000	49,152/49,152	> 2,000,000	x86: redis.cluster.xu1.large.r2.512 Arm: redis.cluster.au1.large.r2.512
1024	1024	128	1,280,000 / 6,400,000	98,304/98,304	> 2,000,000	x86: redis.cluster.xu1.large.r2.1024 Arm: redis.cluster.au1.large.r2.1024

1.4.3 Memcached Instance Specifications

This section describes DCS Memcached instance specifications, including the total memory, available memory, maximum number of connections allowed, maximum/assured bandwidth, and reference performance.

Maximum connections: The maximum number of connections allowed is the maximum number of clients that can be connected to an instance. To check the number of connections to an instance, view the **Connected Clients** metric.

QPS represents queries per second, which is the number of commands processed per second.

 **NOTE**

DCS Memcached instances are available in single-node and master/standby types.

Single-Node Instances

For each single-node DCS Memcached instance, the available memory is less than the total memory because some memory is reserved for system overheads, as shown in [Table 1-11](#).

Table 1-11 Specifications of single-node DCS Memcached instances

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)
2	1.5	5000/50,000	42/128	50,000
4	3.2	5000/50,000	64/192	100,000
8	6.8	5000/50,000	64/192	100,000
16	13.6	5000/50,000	85/256	100,000
32	27.2	5000/50,000	85/256	100,000
64	58.2	5000/50,000	128/384	100,000

Master/Standby Instances

For each master/standby DCS Memcached instance, the available memory is less than the total memory because some memory is reserved for data persistence, as shown in [Table 1-12](#). The available memory of a master/standby instance can be adjusted to support background tasks such as data persistence and master/standby synchronization.

Table 1-12 Specifications of master/standby DCS Memcached instances

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)
2	1.5	5000/50,000	42/128	50,000

Total Memory (GB)	Available Memory (GB)	Max. Connections (Default/Limit) (Count)	Assured/Maximum Bandwidth (Mbit/s)	Reference Performance (QPS)
4	3.2	5000/50,000	64/192	100,000
8	6.8	5000/50,000	64/192	100,000
16	13.6	5000/50,000	85/256	100,000
32	27.2	5000/50,000	85/256	100,000
64	58.2	5000/50,000	128/384	100,000

1.5 Command Compatibility

1.5.1 Redis 3.0 Commands

DCS for Redis 3.0 is developed based on Redis 3.0.7 and is compatible with open-source protocols and commands.

This section describes DCS for Redis 3.0's compatibility with Redis commands, including supported commands, disabled commands, unsupported scripts and commands of later Redis versions, and restrictions on command usage. For more information about the command syntax, visit the [Redis official website](#).

DCS for Redis instances support most Redis commands, which are listed in [Commands Supported by DCS for Redis 3.0](#). Any client compatible with the Redis protocol can access DCS.

- For security purposes, some Redis commands are disabled in DCS, as listed in [Commands Disabled by DCS for Redis 3.0](#).
- Some Redis commands are supported by cluster DCS instances for multi-key operations in the same slot. For details, see [Command Restrictions for Cluster Instances](#).
- Some Redis commands have usage restrictions, which are described in [Other Command Usage Restrictions](#).

Commands Supported by DCS for Redis 3.0

The following lists commands supported by DCS for Redis 3.0.

 NOTE

- Commands available since later Redis versions are not supported by earlier-version instances. Run a command on redis-cli to check whether it is supported by DCS for Redis. If the message "(error) ERR unknown command" is returned, the command is not supported.
- The following commands listed in the tables are not supported by Proxy Cluster instances:
 - **List** group: **BLPOP**, **BRPOP**, and **BRPOPLRUSH**
 - **CLIENT** commands in the **Server** group: **CLIENT KILL**, **CLIENT GETNAME**, **CLIENT LIST**, **CLIENT SETNAME**, **CLIENT PAUSE**, and **CLIENT REPLY**.
 - **Server** group: **MONITOR**
 - **Key** group: **RANDOMKE** (for old Proxy Cluster instances)

Table 1-13 Commands supported by DCS Redis 3.0 instances 1

Keys	String	Hash	List	Set	Sorted Set	Server
DEL	APPEND	HDEL	BLPOP	SADD	ZADD	FLUSHALL
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD	FLUSHDB
EXISTS	BITOP	HGET	BRPOPLRUSH	SDIFF	ZCOUNT	DBSIZE
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY	TIME
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	KEYS
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	CLIENT KILL
RANDOMKEY	GETRANGE	HMGSET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	CLIENT LIST
RENAME	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYSCORE	CLIENT GETNAME
RENAME NX	INCR	HSET	LREM	SPOP	ZREVRANGE	CLIENT SETNAME
RESTORE	INCRBY	HSETNX	LSET	SRANDMEMBER	ZREVRANGEBYSCORE	CONFIG GET
SORT	INCRBYFLOAT	HVALS	LTRIM	SREM	ZREVRANK	MONITOR

Keys	String	Hash	List	Set	Sorted Set	Server
TTL	MGET	HSCAN	RPOP	SUNION	ZSCORE	SLOWLOG
TYPE	MSET	-	RPOPLPU	SUNIONSTORE	ZUNIONSTORE	ROLE
SCAN	MSETNX	-	RPOPLPUSH	SSCAN	ZINTERSTORE	-
OBJECT	PSETEX	-	RPUX	-	ZSCAN	-
-	SET	-	RPUX	-	ZRANGEBYLEX	-
-	SETRBIT	-	-	-	-	-
-	SETEX	-	-	-	-	-
-	SETNX	-	-	-	-	-
-	SETRANGE	-	-	-	-	-
-	STRLEN	-	-	-	-	-

Table 1-14 Commands supported by DCS Redis 3.0 instances 2

HyperLogLog	Pub/Sub	Transactions	Connection	Scripting	Geo
PFADD	PSUBSCRIBE	DISCARD	AUTH	EVAL	GEOADD
PFCOUNT	PUBLISH	EXEC	ECHO	EVALSHA	GEOHASH
PFMERGE	PUBSUB	MULTI	PING	SCRIPT EXISTS	GEOPOS
-	PUNSUBSCRIBE	UNWATCH	QUIT	SCRIPT FLUSH	GEODIST
-	SUBSCRIBE	WATCH	SELECT	SCRIPT KILL	GEORADIUS
-	UNSUBSCRIBE	-	-	SCRIPT LOAD	GEORADIUSBYMEMBER

Commands Disabled by DCS for Redis 3.0

The following lists commands disabled by DCS for Redis 3.0.

Table 1-15 Redis commands disabled in single-node and master/standby Redis 3.0 instances

Keys	Server
MIGRATE	SLAVEOF
-	SHUTDOWN
-	LASTSAVE
-	DEBUG commands
-	COMMAND
-	SAVE
-	BGSAVE
-	BGREWRITEAOF

Table 1-16 Redis commands disabled in Proxy Cluster Redis 3.0 instances

Keys	Server	List	Transactions	Connection	Cluster	codis
MIGRATE	SLAVEOF	BLPOP	DISCARD	SELECT	CLUSTER	TIME
MOVE	SHUTDOWN	BRPOP	EXEC	-	-	SLOTSINFO
-	LASTSAVE	BRPOPLPUSH	MULTI	-	-	SLOTSDEL
-	DEBUG commands	-	UNWATCH	-	-	SLOTSMGRTSLOT
-	COMMAND	-	WATCH	-	-	SLOTSMGRTONE
-	SAVE	-	-	-	-	SLOTSCHECK
-	BGSAVE	-	-	-	-	SLOTSMGRTTAGSLOT
-	BGREWRITEAOF	-	-	-	-	SLOTSMGRTTAGONE
-	SYNC	-	-	-	-	-
-	PSYNC	-	-	-	-	-

Keys	Server	List	Transactions	Connections	Cluster	codis
-	MONITOR	-	-	-	-	-
-	CLIENT commands	-	-	-	-	-
-	OBJECT	-	-	-	-	-
-	ROLE	-	-	-	-	-

1.5.2 Redis 4.0 Commands

DCS for Redis 4.0 is developed based on Redis 4.0.14 and is compatible with open-source protocols and commands.

This section describes DCS for Redis 4.0's compatibility with Redis commands, including supported and disabled commands. For more information about the command syntax, visit the [Redis official website](#).

DCS for Redis instances support most Redis commands, which are listed in [Commands Supported by DCS for Redis 4.0](#). Any client compatible with the Redis protocol can access DCS.

- For security purposes, some Redis commands are disabled in DCS, as listed in [Commands Disabled by DCS for Redis 4.0](#).
- Some Redis commands are supported by cluster DCS instances for multi-key operations in the same slot. For details, see [Command Restrictions for Cluster Instances](#).
- Some Redis commands have usage restrictions, which are described in [Other Command Usage Restrictions](#).

Commands Supported by DCS for Redis 4.0

[Table 1-17](#) and [Table 1-18](#) list the Redis commands supported by single-node, master/standby, and Redis Cluster DCS Redis 4.0 instances.

NOTE

- Commands available since later Redis versions are not supported by earlier-version instances. Run a command on `redis-cli` to check whether it is supported by DCS for Redis. If the message "(error) ERR unknown command" is returned, the command is not supported.
- For DCS Redis 4.0 instances in the Redis Cluster mode, ensure that all commands in a pipeline are executed on the same shard.

Table 1-17 Commands supported by single-node, master/standby, and Redis Cluster DCS Redis 4.0 instances (1)

Keys	String	Hash	List	Set	Sorted Set	Server
DEL	APPEND	HDEL	BLPOP	SADD	ZADD	FLUSHALL
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD	FLUSHDB
EXISTS	BITOP	HGET	BRPOPLRUSH	SDIFF	ZCOUNT	DBSIZE
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY	TIME
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	KEYS
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	CLIENT KILL
RANDOMKEY	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGE BYRANK	CLIENT LIST
RENAME	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGE BYCORE	CLIENT GETNAME
RENAME NX	INCR	HSET	LREM	SPOP	ZREVRANGE	CLIENT SETNAME
RESTORE	INCRBY	HSETNX	LSET	SRANDMEMBER	ZREVRANGE BYSCORE	CONFIG GET
SORT	INCRBYFLOAT	HVALS	LTRIM	SREM	ZREVRANK	MONITOR
TTL	MGET	HSCAN	RPOP	SUNION	ZSCORE	SLOWLOG
TYPE	MSET	HSTRLEN	RPOPLPUSH	SUNIONSTORE	ZUNIONSTORE	ROLE
SCAN	MSETNX	HLEN	RPOPLPUSH	SSCAN	ZINTERSTORE	SWAPDB
OBJECT	PSETEX	-	RPUSH	SPOP	ZSCAN	MEMORY
PEXPIRE	SET	-	RPUSHX	-	ZRANGEBYLEX	CONFIG
PEXPIREAT	SETBIT	-	LPUSH	-	ZLEXCOUNT	-

Keys	String	Hash	List	Set	Sorted Set	Server
-	SETEX	-	-	-	ZREMRANGE BYSCORE	-
-	SETNX	-	-	-	ZREM	-
-	SETRANGE	-	-	-	-	-
-	STRLEN	-	-	-	-	-
-	BITFIELD	-	-	-	-	-

Table 1-18 Commands supported by single-node, master/standby, and Redis Cluster DCS Redis 4.0 instances (2)

HyperLogLog	Pub/Sub	Transactions	Connection	Scripting	Geo
PFADD	PSUBSCRIBE	DISCARD	AUTH	EVAL	GEOADD
PFCOUNT	PUBLISH	EXEC	ECHO	EVALSHA	GEOHASH
PFMERGE	PUBSUB	MULTI	PING	SCRIPT EXISTS	GEOPOS
-	PUNSUBSCRIBE	UNWATCH	QUIT	SCRIPT FLUSH	GEODIST
-	SUBSCRIBE	WATCH	SELECT	SCRIPT KILL	GEORADIUS
-	UNSUBSCRIBE	-	-	SCRIPT LOAD	GEORADIUSBYMEMBER

Commands Disabled by DCS for Redis 4.0

The following lists commands disabled by DCS for Redis 4.0.

Table 1-19 Redis commands disabled in single-node and master/standby Redis 4.0 instances

Keys	Server
MIGRATE	SLAVEOF
-	SHUTDOWN
-	LASTSAVE

Keys	Server
-	DEBUG commands
-	COMMAND
-	SAVE
-	BGSAVE
-	BGREWRITEAOF
-	SYNC
-	PSYNC

Table 1-20 Redis commands disabled in Redis Cluster Redis 4.0 instances

Keys	Server	Cluster
MIGRATE	SLAVEOF	CLUSTER MEET
-	SHUTDOWN	CLUSTER FLUSHSLOTS
-	LASTSAVE	CLUSTER ADDSLOTS
-	DEBUG commands	CLUSTER DELSLOTS
-	COMMAND	CLUSTER SETSLOT
-	SAVE	CLUSTER BUMPEPOCH
-	BGSAVE	CLUSTER SAVECONFIG
-	BGREWRITEAOF	CLUSTER FORGET
-	SYNC	CLUSTER REPLICATE
-	PSYNC	CLUSTER COUNT-FAILURE-REPORTS
-	-	CLUSTER FAILOVER
-	-	CLUSTER SET-CONFIG-EPOCH
-	-	CLUSTER RESET

1.5.3 Redis 5.0 Commands

DCS for Redis 5.0 is developed based on Redis 5.0.9 and is compatible with open-source protocols and commands.

This section describes DCS for Redis 5.0's compatibility with Redis commands, including supported and disabled commands. For more information about the command syntax, visit the [Redis official website](#).

DCS for Redis instances support most Redis commands. Any client compatible with the Redis protocol can access DCS.

- For security purposes, some Redis commands are disabled in DCS, as listed in [Commands Disabled by DCS for Redis 5.0](#).
- Some Redis commands are supported by cluster DCS instances for multi-key operations in the same slot. For details, see [Command Restrictions for Cluster Instances](#).
- Some Redis commands have usage restrictions, which are described in [Other Command Usage Restrictions](#).

Commands Supported by DCS for Redis 5.0

- [Table 1-21](#) and [Table 1-22](#) list commands supported by single-node, master/standby, and Redis Cluster DCS for Redis 5.0.

NOTE

- Commands available since later Redis versions are not supported by earlier-version instances. Run a command on redis-cli to check whether it is supported by DCS for Redis. If the message "(error) ERR unknown command" is returned, the command is not supported.
- For DCS Redis 5.0 instances in the Redis Cluster mode, ensure that all commands in a pipeline are executed on the same shard.

Table 1-21 Commands supported by single-node, master/standby, and Redis Cluster DCS Redis 5.0 instances (1)

Keys	String	Hash	List	Set	Sorted Set	Server
DEL	APPEND	HDEL	BLPOP	SADD	ZADD	FLUSHALL
DUMP	BITCOUNT	HEXISTS	BRPOP	SCARD	ZCARD	FLUSHDB
EXISTS	BITOP	HGET	BRPOPLRUSH	SDIFF	ZCOUNT	DBSIZE
EXPIRE	BITPOS	HGETALL	LINDEX	SDIFFSTORE	ZINCRBY	TIME
MOVE	DECR	HINCRBY	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	HINCRBYFLOAT	LLEN	SINTERSTORE	ZRANGEBYSCORE	KEYS
PTTL	GET	HKEYS	LPOP	SISMEMBER	ZRANK	CLIENT KILL
RANDOMKEY	GETRANGE	HMGET	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	CLIENT LIST
RENAME	GETSET	HMSET	LRANGE	SMOVE	ZREMRANGEBYSCORE	CLIENT GETNAME

Keys	String	Hash	List	Set	Sorted Set	Server
RENAME NX	INCR	HSET	LREM	SPOP	ZREVRANGE	CLIENT SETNAME
RESTOR E	INCRBY	HSETN X	LSET	SRAND MEMBE R	ZREVRANGE BYSCORE	CONFIG GET
SORT	INCRBY FLOAT	HVALS	LTRIM	SREM	ZREVRANK	MONITOR
TTL	MGET	HSCAN	RPOP	SUNION	ZSCORE	SLOWLOG
TYPE	MSET	HSTRLE N	RPOPL PU	SUNION STORE	ZUNIONSTO RE	ROLE
SCAN	MSETN X	HLEN	RPOPL PUSH	SSCAN	ZINTERSTOR E	SWAPDB
OBJECT	PSETEX	-	RPUSH	SPOP	ZSCAN	MEMORY
PEXPIRE AT	SET	-	RPUSH X	-	ZRANGEBYL EX	CONFIG
PEXPIRE	SETBIT	-	LPUSH	-	ZLEXCOUNT	-
-	SETEX	-	-	-	ZPOPMIN	-
-	SETNX	-	-	-	ZPOPMAX	-
-	SETRAN GE	-	-	-	ZREMRANGE BYSCORE	-
-	STRLEN	-	-	-	ZREM	-
-	BITFIEL D	-	-	-	-	-

Table 1-22 Commands supported by single-node, master/standby, and Redis Cluster DCS Redis 5.0 instances (2)

HyperLo glog	Pub/Su b	Transac tions	Connec tion	Scriptin g	Geo	Stream
PFADD	PSUBSC RIBE	DISCAR D	AUTH	EVAL	GEOADD	XACK
PFCOUN T	PUBLIS H	EXEC	ECHO	EVALSH A	GEOHASH	XADD
PFMERG E	PUBSUB	MULTI	PING	SCRIPT EXISTS	GEOPOS	XCLAIM
-	PUNSU BSCRIBE	UNWAT CH	QUIT	SCRIPT FLUSH	GEODIST	XDEL

HyperLoglog	Pub/Sub	Transactions	Connection	Scripting	Geo	Stream
-	SUBSCRIBE	WATCH	SELECT	SCRIPT KILL	GEORADIUS	XGROUP
-	UNSUBSCRIBE	-	-	SCRIPT LOAD	GEORADIUS BYMEMBER	XINFO
-	-	-	-	-	-	XLEN
-	-	-	-	-	-	XPENDING
-	-	-	-	-	-	XRANGE
-	-	-	-	-	-	XREAD
-	-	-	-	-	-	XREADGROUP
-	-	-	-	-	-	XREVRANGE
-	-	-	-	-	-	XTRIM

Commands Disabled by DCS for Redis 5.0

The following lists commands disabled by DCS for Redis 5.0.

Table 1-23 Redis commands disabled in single-node and master/standby Redis 5.0 instances

Keys	Server
MIGRATE	SLAVEOF
-	SHUTDOWN
-	LASTSAVE
-	DEBUG commands
-	COMMAND
-	SAVE
-	BGSAVE
-	BGREWRITEAOF
-	SYNC
-	PSYNC

Table 1-24 Redis commands disabled in Redis Cluster Redis 5.0 instances

Keys	Server	Cluster
MIGRATE	SLAVEOF	CLUSTER MEET
-	SHUTDOWN	CLUSTER FLUSHSLOTS
-	LASTSAVE	CLUSTER ADDSLOTS
-	DEBUG commands	CLUSTER DELSLOTS
-	COMMAND	CLUSTER SETSLOT
-	SAVE	CLUSTER BUMPEPOCH
-	BGSAVE	CLUSTER SAVECONFIG
-	BGREWRITEAOF	CLUSTER FORGET
-	SYNC	CLUSTER REPLICATE
-	PSYNC	CLUSTER COUNT-FAILURE-REPORTS
-	-	CLUSTER FAILOVER
-	-	CLUSTER SET-CONFIG-EPOCH
-	-	CLUSTER RESET

1.5.4 Web CLI Commands

Web CLI is a command line tool provided on the DCS console. This section describes Web CLI's compatibility with Redis commands, including supported and disabled commands. For details about the command syntax, visit the [Redis official website](#).

Currently, only DCS for Redis 4.0 and 5.0 support Web CLI.

NOTE

- Keys and values cannot contain spaces.
- If the value is empty, **nil** is returned after the **GET** command is executed.

Commands Supported by Web CLI

The following lists the commands supported when you use Web CLI.

Table 1-25 Commands supported by Web CLI (1)

Keys	String	List	Set	Sorted Set	Server
DEL	APPEND	RPUSH	SADD	ZADD	FLUSHALL

Keys	String	List	Set	Sorted Set	Server
OBJECT	BITCOUNT	RPUSHX	SCARD	ZCARD	FLUSHDB
EXISTS	BITOP	BRPOPLRUSH	SDIFF	ZCOUNT	DBSIZE
EXPIRE	BITPOS	LINDEX	SDIFFSTORE	ZINCRBY	TIME
MOVE	DECR	LINSERT	SINTER	ZRANGE	INFO
PERSIST	DECRBY	LLEN	SINTERSTORE	ZRANGEBYSCORE	CLIENT KILL
PTTL	GET	LPOP	SISMEMBER	ZRANK	CLIENT LIST
RANDOM KEY	GETRANGE	LPUSHX	SMEMBERS	ZREMRANGEBYRANK	CLIENT GETNAME
RENAME	GETSET	LRANGE	SMOVE	ZREMRANGEBYSCORE	CLIENT SETNAME
RENAMENX	INCR	LRM	SPOP	ZREVRANGE	CONFIG GET
SCAN	INCRBY	LSET	SRANDMEMBER	ZREVRANGEBYSCORE	MONITOR
SORT	INCRBYFLOAT	LTRIM	SREM	ZREVRANK	SLOWLOG
TTL	MGET	RPOP	SUNION	ZSCORE	ROLE
TYPE	MSET	RPOPLPU	SUNIONSTORE	ZUNIONSTORE	SWAPDB
-	MSETNX	RPOPLPUSH	SSCAN	ZINTERSTORE	MEMORY
-	PSETEX	-	SPOP	ZSCAN	-
-	SET	-	-	ZRANGEBYLEX	-
-	SETBIT	-	-	ZLEXCOUNT	-
-	SETEX	-	-	-	-
-	SETNX	-	-	-	-
-	SETRANGE	-	-	-	-
-	STRLEN	-	-	-	-
-	BITFIELD	-	-	-	-

Table 1-26 Commands supported by Web CLI (2)

Hash	HyperLoglog	Connection	Scripting	Geo
HDEL	PFADD	AUTH	EVAL	GEOADD
HEXISTS	PFCOUNT	ECHO	EVALSHA	GEOHASH
HGET	PFMERGE	PING	SCRIPT EXISTS	GEOPOS
HGETALL	-	QUIT	SCRIPT FLUSH	GEODIST
HINCRBY	-	-	SCRIPT KILL	GEORADIUS
HINCRBYFLOAT	-	-	SCRIPT LOAD	GEORADIUSBYMEMBER
HKEYS	-	-	-	-
HMGET	-	-	-	-
HMSET	-	-	-	-
HSET	-	-	-	-
HSETNX	-	-	-	-
HVALS	-	-	-	-
HSCAN	-	-	-	-
HSTRLEN	-	-	-	-

Commands Disabled in Web CLI

The following lists the commands disabled when you use Web CLI.

Table 1-27 Redis commands disabled in Web CLI for single-node and master/standby instances (1)

Keys	Server	Transactions	Pub/Sub
MIGRATE	SLAVEOF	UNWATCH	PUBLISH
WAIT	SHUTDOWN	REPLICAOF	PUBSUB
DUMP	DEBUG commands	DISCARD	PUNSUBSCRIBE
RESTORE	CONFIG SET	EXEC	SUBSCRIBE
-	CONFIG REWRITE	MULTI	UNSUBSCRIBE
-	CONFIG RESETSTAT	WATCH	-
-	SAVE	-	-

Keys	Server	Transactions	Pub/Sub
-	BGSAVE	-	-
-	BGREWRITEAOF	-	-
-	COMMAND	-	-
-	KEYS	-	-
-	MONITOR	-	-
-	SYNC	-	-
-	PSYNC	-	-
-	ACL	-	-

Table 1-28 Redis commands disabled in Web CLI for single-node and master/standby instances (2)

List	Connection	Sorted Set
BLPOP	SELECT	BZPOPMAX
BRPOP	-	BZPOPMIN
BLMOVE	-	BZMPOP
BRPOPLPUSH	-	-
BLMPOP	-	-

Table 1-29 Redis commands disabled in Web CLI for Redis Cluster instances (1)

Keys	Server	Transactions	Cluster
MIGRATE	SLAVEOF	UNWATCH	CLUSTER MEET
WAIT	SHUTDOWN	REPLICAOF	CLUSTER FLUSHSLOTS
DUMP	DEBUG commands	DISCARD	CLUSTER ADDSLOTS
RESTORE	CONFIG SET	EXEC	CLUSTER DELSLOTS
-	CONFIG REWRITE	MULTI	CLUSTER SETSLOT
-	CONFIG RESETSTAT	WATCH	CLUSTER BUMPEPOCH
-	SAVE	-	CLUSTER SAVECONFIG
-	BGSAVE	-	CLUSTER FORGET
-	BGREWRITEAOF	-	CLUSTER REPLICATE

Keys	Server	Transactions	Cluster
-	COMMAND	-	CLUSTER COUNT-FAILURE-REPORTS
-	KEYS	-	CLUSTER FAILOVER
-	MONITOR	-	CLUSTER SET-CONFIG-EPOCH
-	SYNC	-	CLUSTER RESET
-	PSYNC	-	-
-	ACL	-	-

Table 1-30 Redis commands disabled in Web CLI for Redis Cluster instances (2)

Pub/Sub	List	Connection	Sorted Set
PSUBSCRIBE	BLPOP	SELECT	BZPOPMAX
PUBLISH	BRPOP	-	BZPOPMIN
PUBSUB	BLMOVE	-	BZMPOP
PUNSUBSCRIBE	BRPOPLPUSH	-	-
SUBSCRIBE	BLMPOP	-	-
UNSUBSCRIBE	-	-	-

1.5.5 Memcached Commands

Memcached supports the TCP-based text protocol and binary protocol. Any clients compatible with a Memcached protocol can access DCS instances.

Memcached Text Protocol

The Memcached text protocol uses ASCII text to transfer commands, which helps you compile clients and debug problems. DCS Memcached instances can even be directly connected using Telnet.

Compared with the Memcached binary protocol, the Memcached text protocol is compatible with more open-source clients, but the text protocol does not support authentication.

 **NOTE**

Clients can use the Memcached text protocol to access DCS Memcached instances only if password-free access is enabled. Password-free access means that access to DCS Memcached instances will not be username- and password-protected, and any Memcached clients that satisfy security group rules in the same VPC can access the instances. Enabling password-free access poses security risks. Exercise caution when enabling password-free access.

Table 1-31 lists the commands supported by the Memcached text protocol and describes whether these commands are supported by DCS Memcached instances.

Table 1-31 Commands supported by the Memcached text protocol

Command	Function	Supported by DCS
add	Adding data	Yes
set	Sets data, including adding or modifying data.	Yes
replace	Replaces data.	Yes
append	Adds data after the value of the specified key.	Yes
prepend	Adds data before the value of a specified key.	Yes
cas	Checks and set data.	Yes
get	Queries data.	Yes
gets	Queries data details.	Yes
delete	Deletes data.	Yes
incr	Adds the specified amount to the requested counter.	Yes
decr	Removes the specified amount to the requested counter.	Yes
touch	Updates the expiration time of existing data.	Yes
quit	Closes the connection.	Yes
flush_all	Clearing DCS instance data NOTE The value of the delay option (if any) must be 0 .	Yes
version	Queries Memcached version information.	Yes

Command	Function	Supported by DCS
stats	Manages operation statistics. NOTE Currently, only basic statistics can be queried. Commands on optional parameters cannot be queried.	Yes
cache_memlimit	Adjusts the cache memory limit.	No
slabs	Queries usage of internal storage structures.	No
lru	Manages policies of deleting expired data.	No
lru_crawler	Manages threads of deleting expired data.	No
verbosity	Sets the verbosity level of the logging output.	No
watch	Inspects what's going on internally.	No

Memcached Binary Protocol

The Memcached binary protocol encodes commands and operations into specific structures before sending them. Commands are represented by predefined character strings.

The Memcached binary protocol provides more features but fewer clients than the Memcached text protocol. The Memcached binary protocol is more secure than the Memcached text protocol as it additionally supports simple authentication and security layer (SASL) authentication.

Table 1-32 lists the commands supported by the Memcached binary protocol and describes whether these commands are supported by DCS Memcached instances.

Table 1-32 Commands supported by the Memcached binary protocol

Command Code	Command	Function	Supported by DCS
0x00	GET	Queries data.	Yes
0x01	SET	Sets data, including adding or modifying data.	Yes
0x02	ADD	Adding data	Yes
0x03	REPLACE	Replaces data.	Yes
0x04	DELETE	Deletes data.	Yes
0x05	INCREMENT	Adds the specified amount to the requested counter.	Yes

Command Code	Command	Function	Supported by DCS
0x06	DECREMENT	Removes the specified amount to the requested counter.	Yes
0x07	QUIT	Closes the connection.	Yes
0x08	FLUSH	Clearing DCS instance data NOTE The value of the delay option (if any) must be 0 .	Yes
0x09	GETQ	Queries data. The client will not receive any response in case of failure.	Yes
0x0a	NOOP	No-operation instruction, equivalent to ping.	Yes
0x0b	VERSION	Queries Memcached version information.	Yes
0x0c	GETK	Queries data and adds a key into the response packet.	Yes
0x0d	GETKQ	Queries data and returns a key. The client will not receive any response in case of failure.	Yes
0x0e	APPEND	Adds data after the value of the specified key.	Yes
0x0f	PREPEND	Adds data before the value of a specified key.	Yes
0x10	STAT	Queries statistics of DCS Memcached instances. NOTE Currently, only basic statistics can be queried. Commands on optional parameters cannot be queried.	Yes
0x11	SETQ	Sets data, including adding or modifying data. The SETQ command only returns a response on failures. The client will not receive any response in the case of success.	Yes
0x12	ADDQ	Adds data. The client will not receive any response in the case of success.	Yes
0x13	REPLACEQ	Replaces data. The client will not receive any response in the case of success.	Yes

Command Code	Command	Function	Supported by DCS
0x14	DELETEQ	Deletes data. The client will not receive any response in the case of success.	Yes
0x15	INCREMENT Q	Adds the specified amount to the requested counter. The client will not receive any response in the case of success.	Yes
0x16	DECREMENT Q	Removes the specified amount to the requested counter. The client will not receive any response in the case of success.	Yes
0x17	QUITQ	Closes the connection.	Yes
0x18	FLUSHQ	Clears data and returns no information. NOTE The value of the delay option (if any) must be 0.	Yes
0x19	APPENDQ	Adds data after the value of the specified key. The client will not receive any response in the case of success.	Yes
0x1a	PREPENDQ	Adds data before the value of a specified key. The client will not receive any response in the case of success.	Yes
0x1c	TOUCH	Updates the expiration time of existing data.	Yes
0x1d	GAT	Queries data and updates the expiration time of existing data.	Yes
0x1e	GATQ	Queries data and returns a key. The client will not receive any response in case of failure.	Yes
0x23	GATK	Queries data, adds a key into the response packet, and updates the expiration time of existing data.	Yes
0x24	GATKQ	Queries data, returns a key, and updates the expiration time of existing data. The client will not receive any response in case of failure.	Yes

Command Code	Command	Function	Supported by DCS
0x20	SASL_LIST_MECHS	Asks the server what SASL authentication mechanisms it supports.	Yes
0x21	SASL_AUTH	Starts SASL authentication.	Yes
0x22	SASL_STEP	Further authentication steps are required.	Yes

1.5.6 Command Restrictions for Cluster Instances

Some Redis commands are supported by cluster DCS instances for multi-key operations in the same slot. For details, see [Table 1-33](#).

Table 1-33 Redis commands restricted in cluster DCS instances.

Category	Description
Set	
SINTER	Returns the members of the set resulting from the intersection of all the given sets.
SINTERSTORE	Equal to SINTER , but instead of returning the result set, it is stored in <i>destination</i> .
SUNION	Returns the members of the set resulting from the union of all the given sets.
SUNIONSTORE	Equal to SUNION , but instead of returning the result set, it is stored in <i>destination</i> .
SDIFF	Returns the members of the set resulting from the difference between the first set and all the successive sets.
SDIFFSTORE	Equal to SDIFF , but instead of returning the result set, it is stored in <i>destination</i> .
SMOVE	Moves member from the set at source to the set at <i>destination</i> .
Sorted Set	
ZUNIONSTORE	Computes the union of <i>numkeys</i> sorted sets given by the specified keys.
ZINTERSTORE	Computes the intersection of <i>numkeys</i> sorted sets given by the specified keys.
HyperLogLog	

Category	Description
PFCOUNT	Returns the approximated cardinality computed by the HyperLogLog data structure stored at the specified variable.
PFMERGE	Merges multiple HyperLogLog values into a unique value.
Keys	
RENAME	Renames <i>key</i> to <i>newkey</i> .
RENAMENX	Renames <i>key</i> to <i>newkey</i> if <i>newkey</i> does not yet exist.
BITOP	Performs a bitwise operation between multiple keys (containing string values) and stores the result in the destination key.
RPOPLPUSH	Returns and removes the last element (tail) of the list stored at source, and pushes the element at the first element (head) of the list stored at <i>destination</i> .
String	
MSETNX	Merges multiple HyperLogLog values into a unique value.

 **NOTE**

While running commands that take a long time to run, such as **FLUSHALL**, DCS instances may not respond to other commands and may change to the faulty state. After the command finishes executing, the instance will return to normal.

1.5.7 Other Command Usage Restrictions

This section describes restrictions on some Redis commands.

KEYS Command

In case of a large amount of cached data, running the **KEYS** command may block the execution of other commands for a long time or occupy exceptionally large memory. Therefore, when running the **KEYS** command, describe the exact pattern and do not use fuzzy **keys ***. Do not use the **KEYS** command in the production environment. Otherwise, the service running will be affected.

Commands in the Server Group

- While running commands that take a long time to run, such as **FLUSHALL**, DCS instances may not respond to other commands and may change to the faulty state. After the command finishes executing, the instance will return to normal.

- When the **FLUSHDB** or **FLUSHALL** command is run, execution of other service commands may be blocked for a long time in case of a large amount of cached data.

EVAL and EVALSHA Commands

- When the **EVAL** or **EVALSHA** command is run, at least one key must be contained in the command parameter. Otherwise, the error message "ERR eval/evalsha numkeys must be bigger than zero in redis cluster mode" is displayed.
- When the **EVAL** or **EVALSHA** command is run, a cluster DCS Redis instance uses the first key to compute slots. Ensure that the keys to be operated in your code are in the same slot. For details, visit <https://redis.io/commands>.
- For the **EVAL** command:
 - You are advised to learn the Lua script features of Redis before running the **EVAL** command. For details, see <https://redis.io/commands/eval>.
 - The execution timeout time of a Lua script is 5 seconds. Time-consuming statements such as long-time sleep and large loop statements should be avoided.
 - When calling a Lua script, do not use random functions to specify keys. Otherwise, the execution results are inconsistent on the master and standby nodes.

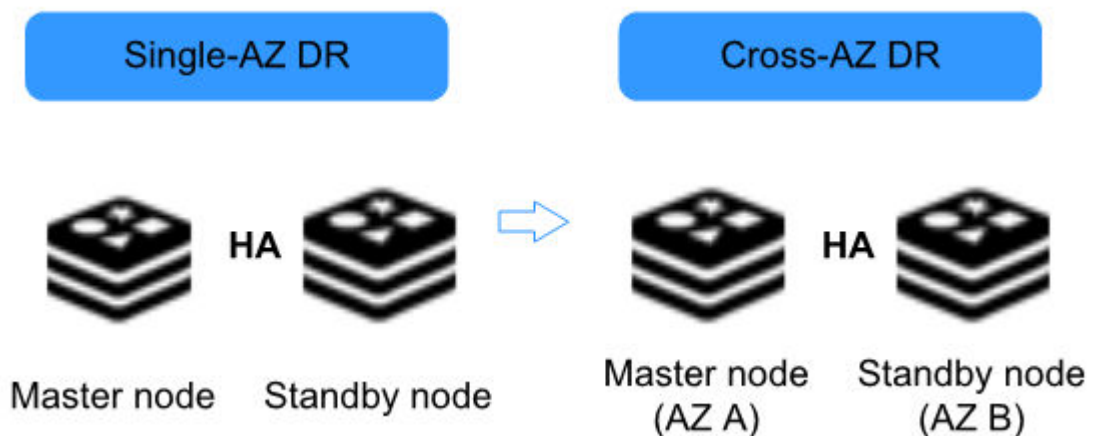
Other Restrictions

- The time limit for executing a Redis command is 15 seconds. To prevent other services from failing, a master/replica switchover will be triggered after the command execution times out.

1.6 HA and DR Policies

Whether you use DCS as the frontend cache or backend data store, DCS is always ready to ensure data reliability and service availability. The following figure shows the evolution of DCS DR architectures.

Figure 1-8 DCS DR architecture evolution



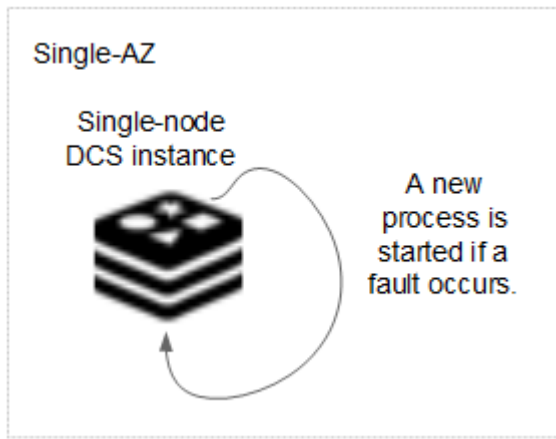
To meet the reliability requirements of your data and services, you can choose to deploy your DCS instance within a single AZ or across AZs.

Single-AZ HA

Single-AZ deployment means deploying an instance within a physical equipment room. DCS provides process/service HA, data persistence, and hot standby DR policies for different types of DCS instances.

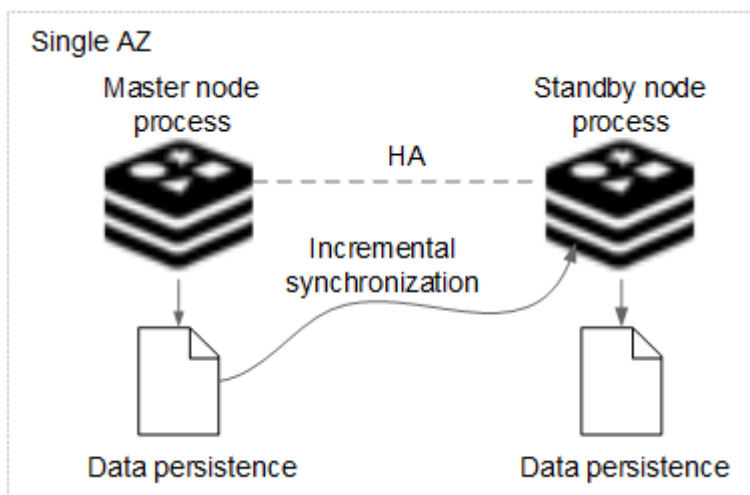
Single-node DCS instance: When DCS detects a process fault, a new process is started to ensure service HA.

Figure 1-9 HA for a single-node DCS instance deployed within an AZ



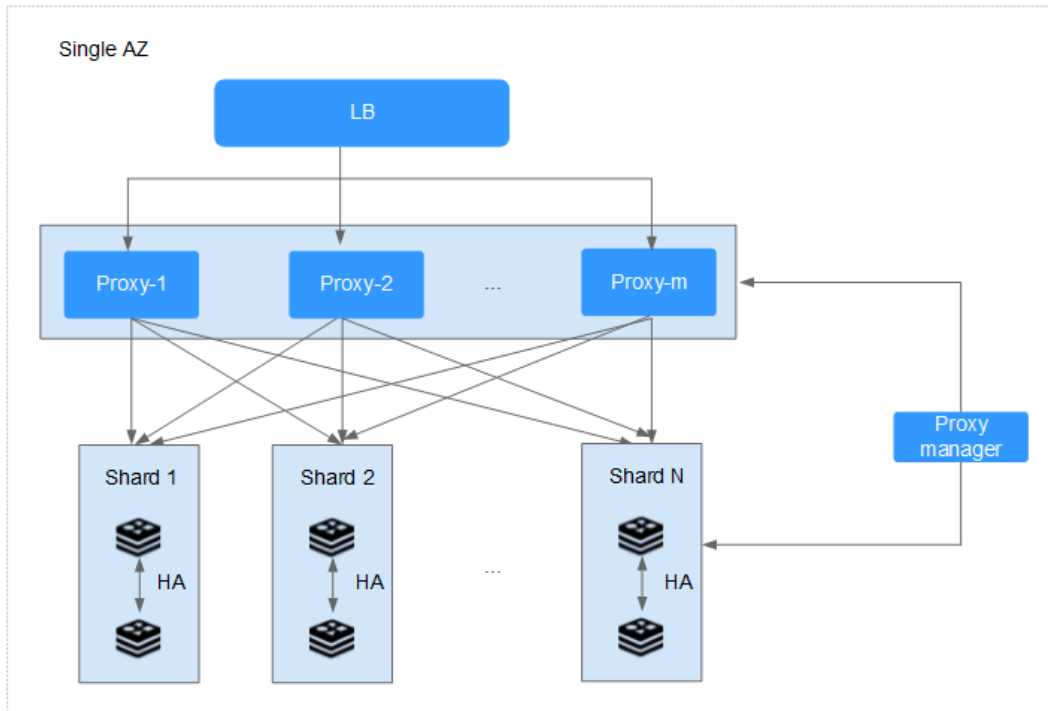
Master/Standby DCS instance: Data is persisted to disk in the master node and incrementally synchronized and persisted to the standby node, achieving hot standby and data persistence.

Figure 1-10 HA for a master/standby DCS instance deployed within an AZ



Cluster DCS instance: Similar to a master/standby instance, data in each shard (instance process) of a cluster instance is synchronized between master and standby nodes and persisted on both nodes.

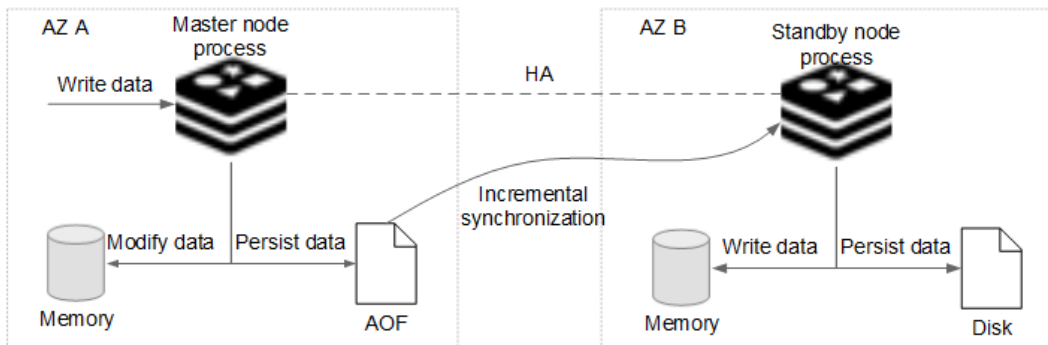
Figure 1-11 HA for a cluster DCS instance deployed within an AZ



Cross-AZ DR

The master and standby nodes of a master/standby or cluster DCS instance can be deployed across AZs (in different equipment rooms). Power supplies and networks of different AZs are physically isolated. When a fault occurs in the AZ where the master node is deployed, the standby node connects to the client and takes over data read and write operations.

Figure 1-12 Cross-AZ deployment of a master/standby DCS instance



NOTE

This mechanism applies in a similar way to a cluster DCS instance. Each shard (process) is deployed across AZs.

When creating a master/standby or cluster DCS instance, select a standby AZ that is different from the primary AZ.

Backup, configuration modification, and password change functions cannot be used during the fault.

1.7 Comparing Redis Versions

When creating a DCS Redis instance, you can select the cache engine version and the instance type.

- **Version**

DCS supports Redis 3.0, 4.0, and 5.0. The following table describes the differences between these versions.

Table 1-34 Differences between Redis versions

Feature	Redis 3.0	Redis 4.0 and Redis 5.0
Instance deployment mode	Based on VMs	Containerized based on physical servers
Time required for creating an instance	3–15 minutes, or 10–30 minutes for cluster instances.	8 seconds
QPS	100,000 QPS per node	100,000 QPS per node
Instance type	Single-node, master/standby, and Proxy Cluster	Single-node, master/standby and Redis Cluster
Instance total memory	Ranges from 2 GB, 4 GB, 8 GB, to 1024 GB.	Regular specifications range from 2 GB, 4 GB, 8 GB, to 1024 GB. Small specifications, such as 128 MB, 256 MB, 512 MB, and 1 GB, are also available for single-node and master/standby instances.
Scale-up or scale-down	Online scale-up and scale-down	Online scale-up and scale-down
Backup and restoration	Supported for master/standby and cluster instances	Supported for master/standby and cluster instances

 **NOTE**

The underlying architectures vary by Redis version. Once a Redis version is chosen, it cannot be changed. For example, you cannot upgrade a DCS Redis 3.0 instance to Redis 4.0 or 5.0. If you require a higher Redis version, create a new instance that meets your requirements and then migrate data from the old instance to the new one.

- Instance type**
 Select from single-node, master/standby, and cluster types. For details about their architectures and application scenarios, see [DCS Instance Types](#).

1.8 Comparing Redis and Memcached

Redis and Memcached are both popular open-source in-memory databases which are easy to use and provide higher performance than relational databases.

How can I select between the two key-value databases?

Memcached is suitable for storing simple data structures, whereas Redis is suitable for storing more complex, larger data that requires persistency.

For details, see the following table.

Table 1-35 Differences between Redis and Memcached

Item	Redis	Memcached
Latency	In-memory database with sub-millisecond latency	In-memory database with sub-millisecond latency
Ease of use	Simple syntax and easy to use	Simple syntax and easy to use
Distributed storage	Horizontal expansion in cluster mode	Supported
Multi-language client	Supports client connections in more than 30 languages including Java, C, and Python.	Supports client connections in more than 10 languages including Java, C, and Python.
Thread/Process	Single-core and single-thread Single-thread communication, avoiding unnecessary context switching and contention Non-blocking I/O (I/O multiplexing) is used to reduce resource consumption when multiple clients are connected.	Multi-thread and scalable The Memcached performance can be improved by increasing the number of CPUs. There is an obvious performance advantage in the scenario where the value of key is great.
Persistent storage	Supported Each write operation (adding, deleting, or modifying data) can be recorded on disk (AOF file).	Supported NOTE Persistence is not supported by open-source Memcached, but is supported by DCS for Memcached.
Data structure	Supports complex data structures such as hash, list, set, and sorted set, catering to various scenarios.	Supports simple strings.
Lua script support	Supported	Not supported

Item	Redis	Memcached
Snapshot backup	Supported Snapshots are generated periodically. Therefore, there is no guarantee that data will not be lost. Redis forks a subprocess to generate snapshots. When there is a large amount of data, the Redis service may be interrupted for a short time.	Not supported
Key value restriction	The value of a key can be up to 1 GB.	1 MB
Multiple databases	Supports up to 256 Redis databases.	Not supported

Based on the preceding comparison, both the Redis and Memcached are easy to use and have high performance. However, Redis and Memcached are different in data structure storage, persistence, backup, migration, and script support. You are advised to select the most appropriate cache engine based on actual application scenarios.

 **NOTE**

Memcached is suitable for caching scenarios of small amount of static data, where data is only read without further computing and processing, for example, HTML code snippets.
Redis has richer data structures and wider application scenarios.

1.9 Comparing DCS and Open-Source Cache Services

DCS supports single-node, master/standby, and cluster instances, ensuring high read/write performance and fast data access. It also supports various instance management operations to facilitate your O&M. With DCS, you only need to focus on the service logic, without concerning about the deployment, monitoring, scaling, security, and fault recovery issues.

DCS is compatible with open-source Redis and Memcached, and can be customized based on your requirements. This renders DCS unique features in addition to the advantages of open-source cache databases.

DCS for Redis vs. Open-Source Redis

Table 1-36 Differences between DCS for Redis and open-source Redis

Feature	Open-Source Redis	DCS for Redis
Service deployment	Requires 0.5 to 2 days to prepare servers.	<ul style="list-style-type: none"> Creates a Redis 3.0 instance in 5 to 15 minutes. Creates a containerized Redis 4.0 or 5.0 instance within 8 seconds.
Version	-	Deeply engaged in the open-source community and supports the latest Redis version. Currently, Redis 3.0, 4.0, and 5.0 are supported.
Security	Network and server safety is the user's responsibility.	<ul style="list-style-type: none"> Network security is ensured using VPCs and security groups. Data reliability is ensured by data replication and scheduled backup.
Performance	-	100,000 QPS per node
Monitoring	Provides only basic statistics.	<p>Provides more than 30 monitoring metrics and customizable alarm threshold and policies.</p> <ul style="list-style-type: none"> Various metrics <ul style="list-style-type: none"> External metrics include the number of commands, concurrent operations, connections, clients, and denied connections. Resource usage metrics include CPU usage, physical memory usage, network input throughput, and network output throughput. Internal metrics include instance capacity usage, as well as the number of keys, expired keys, PubSub channels, PubSub patterns, keyspace hits, and keyspace misses. Custom alarm thresholds and policies for different metrics to help identify service faults.
Backup and restoration	Supported	<ul style="list-style-type: none"> Supports scheduled and manual backup. Backup files can be downloaded. Backup data can be restored on the console.

Feature	Open-Source Redis	DCS for Redis
Parameter management	No visualized parameter management	<ul style="list-style-type: none"> Visualized parameter management is supported on the console. Configuration parameters can be modified online. Data can be accessed and modified on the console.
Scale-up	Interrupts services and involves a complex procedure from modifying the server RAM to modifying Redis memory and restarting the OS and services.	<ul style="list-style-type: none"> Supports online scale-up and scale-down without interrupting services. Specifications can be scaled up or down within the available range based on service requirements.

DCS for Memcached vs. Open-Source Memcached

Table 1-37 Differences between DCS for Memcached and open-source Memcached

Feature	Open-Source Memcached	DCS for Memcached
Service deployment	Requires 0.5 to 2 days to prepare servers.	Creates an instance in 5 to 15 minutes.
Security	Network and server safety is the user's responsibility.	<ul style="list-style-type: none"> Network security is ensured using VPCs and security groups. Data reliability is ensured by data replication and scheduled backup.
Performance	-	100,000 QPS per node

Feature	Open-Source Memcached	DCS for Memcached
Monitoring	Provides only basic statistics.	<p>Provides more than 30 monitoring metrics and customizable alarm threshold and policies.</p> <ul style="list-style-type: none"> • Various metrics <ul style="list-style-type: none"> - External metrics include the number of commands, concurrent operations, connections, clients, and denied connections. - Resource usage metrics include CPU usage, physical memory usage, network input throughput, and network output throughput. - Internal metrics include instance capacity usage, as well as the number of keys, expired keys, PubSub channels, PubSub patterns, keyspace hits, and keyspace misses. • Custom alarm thresholds and policies for different metrics to help identify service faults.
Backup and restoration	Not supported	<ul style="list-style-type: none"> • Supports scheduled and manual backup. • Backup data can be restored on the console.
Visualized maintenance	No visualized parameter management	<ul style="list-style-type: none"> • Visualized parameter management is supported on the console. • Configuration parameters can be modified online.
Scale-up	Interrupts services and involves a complex procedure from modifying the server RAM to modifying Redis memory and restarting the OS and services.	<ul style="list-style-type: none"> • Supports online scale-up without interrupting services. • Specifications can be scaled up or down within the available range based on service requirements.
Data persistence	Not supported	Supported for master/standby instances

1.10 Basic Concepts

DCS Instance

An instance is the minimum resource unit provided by DCS.

You can select the Redis or Memcached cache engine. Instance types can be single-node, master/standby, or cluster. For each instance type, multiple specifications are available.

For details, see [DCS Instance Specifications](#) and [DCS Instance Types](#).

Project

Projects are used to group and isolate OpenStack resources (computing resources, storage resources, and network resources). A project can be a department or a project team. Multiple projects can be created for one account.

Replica

A replica is a node of a DCS instance. No replication indicates that the instance does not have a standby node. Master/Standby replication indicates that the instance has a standby node. For example, a master/standby DCS instance has a master/standby replication. Each node of a cluster DCS Redis instance has a master/standby replication.

Maintenance Time Window

The maintenance time window is the period when the DCS service team upgrade and maintain the instance.

DCS instance maintenance takes place only once a quarter and does not interrupt services. Even so, you are advised to select a time period when the service demand is low.

When creating an instance, you must specify a maintenance time window, which can be modified after the instance is created.

For details, see: [Modifying Maintenance Time Window](#).

Cross-AZ Deployment

Master/Standby instances are deployed across different AZs with physically isolated power supplies and networks. Applications can also be deployed across AZs to achieve HA for both data and applications.

When creating a master/standby or cluster DCS Redis or Memcached instance, you can select a standby AZ for the standby node.

Shard

A shard is a management unit of a cluster DCS Redis instance. Each shard corresponds to a redis-server process. A cluster consists of multiple shards. Each

shard has multiple slots. Data is distributedly stored in the slots. The use of shards increases cache capacity and concurrent connections.

1.11 Permissions Management

If you need to assign different permissions to employees in your enterprise to access your DCS resources, Identity and Access Management (IAM) is a good choice for fine-grained permissions management. IAM provides identity authentication, permissions management, and access control, helping you secure access to your resources.

With IAM, you can use your account to create IAM users, and assign permissions to the users to control their access to specific resources. For example, some software developers in your enterprise need to use DCS resources but should not be allowed to delete DCS instances or perform any other high-risk operations. In this scenario, you can create IAM users for the software developers and grant them only the permissions required for using DCS resources.

If your account does not require individual IAM users for permissions management, skip this section.

DCS Permissions

By default, new IAM users do not have permissions assigned. You need to add a user to one or more groups, and attach permissions policies or roles to these groups. Users inherit permissions from the groups to which they are added and can perform specified operations on cloud services based on the permissions.

DCS is a project-level service deployed and accessed in specific physical regions. To assign DCS permissions to a user group, specify the scope as region-specific projects and select regions for the permissions to take effect. If **All projects** is selected, the permissions will take effect for the user group in all region-specific projects. When accessing DCS, the users need to switch to a region where they have been authorized to use this service.

You can grant users permissions by using roles and policies.

- **Roles:** A type of coarse-grained authorization mechanism that defines permissions related to user responsibilities. This mechanism provides only a limited number of service-level roles for authorization. When using roles to grant permissions, you must also assign other roles on which the permissions depend to take effect. However, roles are not an ideal choice for fine-grained authorization and secure access control.
- **Policies:** A type of fine-grained authorization mechanism that defines permissions required to perform operations on specific cloud resources under certain conditions. This mechanism allows for more flexible policy-based authorization, meeting requirements for secure access control. For example, you can grant DCS users only the permissions for operating DCS instances. Fine-grained policies are based on APIs. The minimum granularity of a policy is API actions. For the API actions supported by DCS, see "Permissions Policies and Supported Actions".

Table 1 lists all the system-defined roles and policies supported by DCS.

Table 1-38 System-defined roles and policies supported by DCS

Role/Policy Name	Description	Type	Dependency
DCS FullAccess	All permissions for DCS. Users granted these permissions can operate and use all DCS instances.	System-defined policy	None
DCS UserAccess	Common user permissions for DCS, excluding permissions for creating, modifying, deleting DCS instances and modifying instance specifications.	System-defined policy	None
DCS ReadOnlyAccess	Read-only permissions for DCS. Users granted these permissions can only view DCS instance data.	System-defined policy	None

 **NOTE**

The **DCS UserAccess** policy is different from the **DCS FullAccess** policy. If you configure both of them, you cannot create, modify, delete, or scale DCS instances because deny statements will take precedence over allowed statements.

Table 2 lists the common operations supported by each system policy of DCS. Please choose proper system policies according to this table.

Table 1-39 Common operations supported by each system policy

Operation	DCS FullAccess	DCS UserAccess	DCS ReadOnlyAccess
Modifying instance configuration parameters	√	√	×
Deleting background tasks	√	√	×
Accessing instances using Web CLI	√	√	×
Modifying instance running status	√	√	×

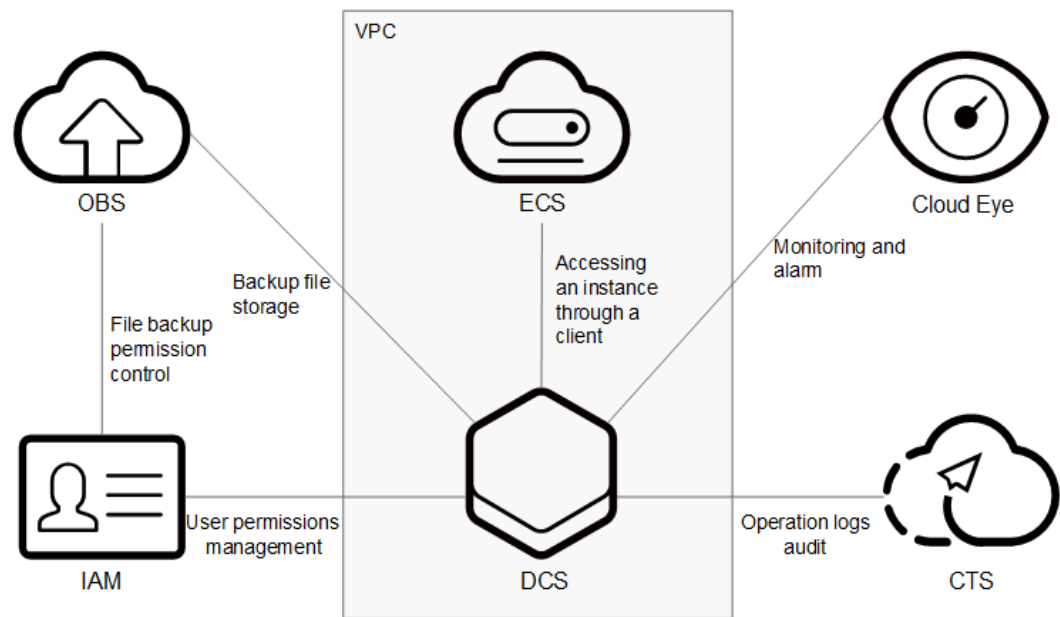
Operation	DCS FullAccess	DCS UserAccess	DCS ReadOnlyAccess
Expanding instance capacity	√	×	×
Changing instance passwords	√	√	×
Modifying DCS instances	√	×	×
Performing a master/standby switchover	√	√	×
Backing up instance data	√	√	×
Analyzing big keys or hot keys	√	√	×
Creating DCS instances	√	×	×
Deleting instance backup files	√	√	×
Upgrading instance version	√	√	×
Restoring instance data	√	√	×
Resetting instance passwords	√	√	×
Migrating instance data	√	√	×
Downloading instance backup data	√	√	×
Deleting DCS instances	√	×	×

Operation	DCS FullAccess	DCS UserAccess	DCS ReadOnlyAccess
Querying instance configuration parameters	√	√	√
Querying instance restoration logs	√	√	√
Querying instance backup logs	√	√	√
Querying DCS instances	√	√	√
Querying instance background tasks	√	√	√
Querying instance upgrade information	√	√	√
Querying all instances	√	√	√
Viewing instance performance metrics	√	√	√

1.12 Related Services

DCS is used together with other services, including VPC, ECS, IAM, Cloud Eye, CTS, and Object Storage Service (OBS).

Figure 1-13 Relationships between DCS and other services



VPC

A VPC is an isolated virtual network environment on the cloud. You can configure IP address ranges, subnets, and security groups in a VPC.

DCS runs in VPCs. The VPC service manages EIPs and bandwidth, and provides security groups. You can configure access rules for security groups to secure the access to DCS.

ECS

An ECS is a cloud server that provides scalable, on-demand computing resources for secure, flexible, and efficient applications.

You can access and manage your DCS instances using an ECS.

IAM

IAM provides identity authentication, permissions management, and access control.

With IAM, you can control access to DCS.

Cloud Eye

Cloud Eye is a secure, scalable, and integrated monitoring service. With Cloud Eye, you can monitor your DCS service and configure alarm rules and notifications.

Cloud Trace Service (CTS)

CTS provides you with a history of operations performed on cloud service resources. With CTS, you can query, audit, and backtrack operations. The traces include the operation requests sent using the management console or open APIs and the results of these requests.

OBS

OBS provides secure, cost-effective storage service using objects as storage units. With OBS, you can store and manage the lifecycle of massive amounts of data.

You can store DCS instance backup files in OBS.

2 Permissions Management

2.1 Creating a User and Granting DCS Permissions

This chapter describes how to use IAM to implement fine-grained permissions control for your DCS resources. With IAM, you can:

- Create IAM users for employees based on your enterprise's organizational structure. Each IAM user will have their own security credentials for accessing DCS resources.
- Grant only the permissions required for users to perform a specific task.
- Entrust an account or cloud service to perform efficient O&M on your DCS resources.

If your account does not need individual IAM users, you may skip over this chapter.

This section describes the procedure for granting the **DCS ReadOnlyAccess** permission (see [Figure 2-1](#)) as an example.

Prerequisites

You are familiar with the permissions (see [Permissions Management](#)) supported by DCS and choose policies or roles according to your requirements. For the permissions of other services, see [Permissions Policies](#).

Process Flow

Figure 2-1 Process of granting DCS permissions



1. Create a user group and grant permissions.
Create a user group on the IAM console, and attach the **DCS ReadOnlyAccess** policy to the group.
2. Create an IAM user.
Create a user on the IAM console and add the user to the group created in **1**.
3. Log in and verify permissions.
Log in to the DCS console by using the newly created user, and verify that the user only has read permissions for DCS.

2.2 DCS Custom Policies

Custom policies can be created to supplement the system-defined policies of DCS. For the actions that can be added for custom policies, see [Permissions Policies and Supported Actions](#).

You can create custom policies in either of the following ways:

- Visual editor: Select cloud services, actions, resources, and request conditions. This does not require knowledge of policy syntax.
- JSON: Edit JSON policies from scratch or based on an existing policy.

For details, see "Creating a Custom Policy". The following section contains examples of common DCS custom policies.

 NOTE

Due to data caching, a policy involving OBS actions will take effect five minutes after it is attached to a user, user group, or project.

Example Custom Policies

- Example 1: Allowing users to delete and restart DCS instances and clear data of an instance

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dcs:instance:delete",
        "dcs:instance:modifyStatus"
      ]
    }
  ]
}
```

- Example 2: Denying DCS instance deletion

A policy with only "Deny" permissions must be used in conjunction with other policies to take effect. If the permissions assigned to a user contain both "Allow" and "Deny", the "Deny" permissions take precedence over the "Allow" permissions.

The following method can be used if you need to assign permissions of the **DCS FullAccess** policy to a user but you want to prevent the user from deleting DCS instances. Create a custom policy for denying DCS instance deletion, and attach both policies to the group to which the user belongs. Then, the user can perform all operations on DCS instances except deleting DCS instances. The following is an example of a deny policy:

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "dcs:instance:delete"
      ]
    }
  ]
}
```

3 Getting Started

3.1 Creating an Instance

3.1.1 Identifying Requirements

Before creating a DCS instance, identify your requirements and complete the following preparations:

1. Decide on the required cache engine.

Choose a cache engine based on service requirements. The cache engine cannot be changed once the instance is created.

- For more information about Redis and Memcached cache engines, see [What Is DCS?](#)
- For more information about the differences between Redis and Memcached, see [Comparing Redis and Memcached](#).

2. Decide on the required cache engine version.

Different Redis versions have different features. For details, see [Comparing Redis Versions](#).

3. Decide on the required instance type.

DCS provides single-node, master/standby, Proxy Cluster, and Redis Cluster types of instances. Each type has its own architecture. For details about the instance architectures, see [DCS Instance Types](#).

4. Decide on the required instance specification.

Each specification specifies the maximum available memory, number of connections, and bandwidth. For details, see [DCS Instance Specifications](#).

5. Decide on the region and whether cross-AZ deployment is required.

Choose a region closest to your application to reduce latency.

A region consists of multiple availability zones (AZs) with physically isolated power supplies and networks. Master/standby and cluster DCS instances can be deployed across AZs.

 **NOTE**

- If a master/standby or cluster DCS instance is deployed across AZs, faults in an AZ do not affect cache nodes in other AZs. This is because when the master node is faulty, the standby cache node will automatically become the master node to provide services. Such deployment achieves better disaster recovery.
 - Deploying a DCS instance across AZs slightly reduces network efficiency compared with deploying an instance within an AZ. Therefore, if a DCS instance is deployed across AZs, synchronization between master and standby cache nodes is slightly less efficient.
6. Decide whether backup policies are required.

Currently, backup policies can be configured only for master/standby and cluster DCS instances. For details about backup and restoration, see [Overview](#).

3.1.2 Preparing the Environment

To access DCS instances through a Virtual Private Cloud (VPC), create a VPC and configure security groups and subnets for it before using DCS. A VPC provides an isolated virtual network environment which you can configure and manage. Using VPCs enhances cloud resource security and simplifies network deployment.

Once you have created a VPC, you can use it for all DCS instances you subsequently create.

Creating a VPC

Step 1 Log in to the management console.

Step 2 Click  in the upper left corner and select a region and a project.

Step 3 Click **Service List**, and choose **Network > Virtual Private Cloud** to launch the VPC console.

Step 4 Click **Apply for VPC**.

Step 5 Create a VPC as prompted, retaining the default values unless otherwise required.

For details about how to create a VPC, see "VPC and Subnet" > "VPC" > "Creating a VPC" in *Virtual Private Cloud User Guide*.

After a VPC is created, a subnet is also created in the subnet. If the VPC needs more subnets, go to [Step 7](#). Otherwise, go to [Step 8](#).

 **NOTE**

- When creating a VPC, **CIDR Block** indicates the IP address range of the VPC. If this parameter is set, the IP addresses of subnets in the VPC must be within the IP address range of the VPC.
- If you create a VPC to provision DCS instances, you do not need to configure the CIDR block for the VPC.

Step 6 In the navigation pane on the left, choose **Virtual Private Cloud > My VPCs**.

Step 7 Click **Create Subnet**. Create a subnet as prompted, retaining the default values unless otherwise required.

For details about how to create a subnet, see "VPC and Subnet" > "Subnet" in *Virtual Private Cloud User Guide*.

Step 8 In the navigation pane on the left, choose **Access Control > Security Groups** and then click **Create Security Group** in the upper right corner of the displayed page. Create a security group as prompted, retaining the default values unless otherwise required.

For details about how to create a security group, see "Security" > "Security Group" > "Creating a Security Group" in *Virtual Private Cloud User Guide*.


----End

3.1.3 Creating a DCS Redis Instance

You can create one or more DCS Redis instances with the required computing capabilities and storage space based on service requirements.

Creating a DCS Redis Instance

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner of the management console and select a region and a project.

Step 3 Click **Create DCS Instance**.

Step 4 Select a region closest to your application to reduce latency and accelerate access.

Step 5 Specify the following instance parameters based on the information collected in [Identifying Requirements](#).

1. **Cache Engine:**

Select **Redis**.

2. **Version:**

Currently, 3.0, 4.0, and 5.0 versions are supported.

 **NOTE**

- When creating a Proxy Cluster instance, you can only select version 3.0.
- When creating a Redis Cluster instance, you can select versions 4.0 or 5.0.

3. Set **Instance Type** to **Single-node, Master/Standby, Proxy Cluster** or **Redis Cluster**.

4. Set **CPU Architecture** to **x86**.

5. Set **Replicas**. The default value is **2** (including the master).

This parameter is displayed only when you select Redis 4.0 or Redis 5.0 and the instance type is master/standby or Redis Cluster.

6. Select an AZ.

If the instance type is master/standby, Proxy Cluster, or Redis Cluster, **Standby AZ** is displayed. Select a standby AZ for the standby node of the instance.

 **NOTE**

- To accelerate access, deploy your instance and your application in the same AZ.
- There are multiple AZs in each region. If resources are insufficient in an AZ, the AZ will be unavailable. In this case, select another AZ.

7. **Instance Specification:**

The remaining quota is displayed on the console.

To apply to increase quota, click **Increase quota** below the specifications.

Step 6 Configure the instance network parameters.

1. For **VPC**, select a created VPC, subnet, and specify the IP address.

You can choose to obtain an automatically assigned IP address or manually specify an IP address that is available in the selected subnet.

For a DCS Redis 4.0 or 5.0 instance, you can specify a port numbering in the range from 1 to 65535. If no port is specified, the default port 6379 will be used. For a DCS Redis 3.0 instance, the port cannot be customized. Port 6379 will be used.

2. Select a security group.

A security group is a set of rules that control access to ECSs. It provides access policies for mutually trusted ECSs with the same security protection requirements in the same VPC.

This parameter can be configured only for instances that use Redis 3.0. DCS for Redis 4.0 and 5.0 are based on VPC endpoints and do not support security groups.

Step 7 Set the instance password.

This password is used for accessing the DCS Redis instance.

 **NOTE**

For security purposes, you must enter an instance-specific password when you are accessing the DCS Redis instance. Keep your instance password secure and change it periodically.

The password must meet the following requirements:

- Cannot be left blank.
- Can contain 8 to 32 characters.
- Must contain at least three of the following character types:
 - Lowercase letters
 - Uppercase letters
 - Digits
 - special characters (^~!@#\$%^&*()-_+=+\\{|};<.>/?)

Step 8 Click **More Settings** to display more configurations, including auto backup.

1. Specify **Name** and **Description**.

The value of **Name** can contain 4 to 64 characters.

2. Choose whether to enable **Auto Backup**.

This parameter is displayed only when the instance type is master/standby or cluster. For more information on how to configure a backup policy, see [Overview](#).

3. Rename critical commands.
Command Renaming is displayed for Redis 4.0 and 5.0. Currently, you can only rename the **COMMAND**, **KEYS**, **FLUSHDB**, **FLUSHALL**, and **HGETALL** commands.
4. Specify the maintenance window.
Choose a window for DCS O&M personnel to perform maintenance on your instance. You will be contacted before any maintenance activities are performed.

Step 9 Click **Create Now**.

The displayed page shows the instance information you have specified.

Step 10 Confirm the instance information and click **Submit**.

Step 11 Return to the **Cache Manager** page to view and manage your DCS instances.

1. Creating a single-node or master/standby DCS Redis 3.0 instance takes 5 to 15 minutes. Creating a cluster DCS Redis 3.0 instance takes 30 minutes.

 **NOTE**

DCS Redis 4.0 and 5.0 instances are containerized and can be created within seconds.

2. After a DCS instance has been successfully created, it enters the **Running** state by default.


----End

3.1.4 Creating a DCS Memcached Instance

You can create one or more DCS Memcached instances with the required computing capabilities and storage space based on service requirements.

Creating a DCS Memcached Instance

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Click **Create DCS Instance**.

Step 5 Select a region closest to your application to reduce latency and accelerate access.

Step 6 Specify the following instance parameters based on the information collected in [Identifying Requirements](#).

1. **Cache Engine:**
Select **Memcached**.
2. **Instance Type:**
Select **Single-node** or **Master/Standby**.
3. Select an AZ.

 **NOTE**

To accelerate access, deploy your instance and your application in the same AZ. To ensure data reliability, deploy them in different AZs.

If the instance type is master/standby, **Standby AZ** is displayed. Select a standby AZ for the standby node of the instance.

4. Specify **Instance Specification**.

The remaining quota is displayed on the console.

To apply to increase quota, click **Increase quota** below the specifications.

Step 7 Configure the instance network parameters.

1. For **VPC**, select a created VPC, subnet, and specify the IP address.

You can choose to obtain an automatically assigned IP address or manually specify an IP address that is available in the selected subnet.

2. Select a security group.

A security group is a set of rules that control access to ECSs. It provides access policies for mutually trusted ECSs with the same security protection requirements in the same VPC.

Step 8 Set the instance password.

- Select **Yes** or **No** for **Password Protected**.

 **NOTE**

- Password-free access carries security risks. Exercise caution when selecting this mode.
- After the instance is created, you can click reset its password.
- If password-free access is disabled, DCS Memcached instances must be accessed using the Memcached binary protocol and through SASL authentication.

- Username required for accessing the new DCS instance. The username must meet the following requirements.

 **NOTE**

This parameter is displayed only if password-protected access is enabled.

- Cannot be left blank.
- Must start with a letter.
- Can contain 1 to 64 characters.
- Must contain only letters, digits, hyphens (-), and underscores (_).

- **Password** and **Confirm Password**: These parameters indicate the password of accessing the DCS Memcached instance, and are displayed only when **Password Protected** is set to **Yes**.

 **NOTE**

For security purposes, if password-free access is disabled, the system prompts you to enter an instance-specific password when you are accessing the DCS Memcached instance. Keep your instance password secure and change it periodically.

Step 9 Click **More Settings** to display more configurations, including backup policy and maintenance window.

1. Specify **Name** and **Description**.
The instance name can contain 4 to 64 characters.
2. Specify backup and restoration policies.
This parameter is displayed only when the instance type is master/standby. For more information on how to configure a backup policy, see [Backing Up and Restoring DCS Instances](#).
3. Specify the maintenance window.
Choose a window for DCS O&M personnel to perform maintenance on your instance. Time windows 22:00–02:00, 02:00–06:00, 06:00–10:00, 10:00–14:00, 14:00–18:00, and 18:00–22:00 are available for selection.

Step 10 Click **Next**.

The displayed page shows the instance information you have specified.

Step 11 Confirm the instance information.**Step 12** After the new DCS instance has been created, return to the **Cache Manager** page to view and manage your DCS instances.

1. It takes 5 to 15 minutes to create a DCS instance.
2. After a DCS instance has been successfully created, it enters the **Running** state by default.

----End

3.2 Accessing an Instance

3.2.1 Accessing a DCS Redis Instance Through redis-cli

Access a DCS Redis instance through redis-cli on an ECS in the same VPC. For more information on how to use other Redis clients, visit <https://redis.io/clients>.

NOTE

- Redis 3.0 does not support port customization and allows only port 6379. For Redis 4.0 and 5.0, you can specify a port or use the default port 6379. The following uses the default port 6379. If you have specified a port, replace 6379 with the actual port.
- **When connecting to a Redis Cluster instance, ensure that -c is added to the command.** Otherwise, the connection will fail.
 - Run the following command to connect to a Redis Cluster instance:

```
./redis-cli -h {dcs_instance_address} -p 6379 -a {password} -c
```
 - Run the following command to connect to a single-node, master/standby, or Proxy Cluster instance:

```
./redis-cli -h {dcs_instance_address} -p 6379 -a {password}
```

For details, see [Step 3](#) and [Step 4](#).

Prerequisites

- The DCS Redis instance you want to access is in the **Running** state.
- An ECS has been created. For more information on how to create ECSs, see the *Elastic Cloud Server User Guide*.

- If the ECS runs the Linux OS, ensure that the GCC compilation environment has been installed on the ECS.

Procedure (Linux)

Step 1 Obtain the IP address and port number of the DCS Redis instance to be accessed.

For details, see [Viewing Details of a DCS Instance](#).

Step 2 Install redis-cli.

The following steps assume that your client is installed on the Linux OS.

1. Log in to the ECS.
2. Run the following command to download the source code package of your Redis client from <http://download.redis.io/releases/redis-5.0.8.tar.gz>:
wget http://download.redis.io/releases/redis-5.0.8.tar.gz
3. Run the following command to decompress the source code package of your Redis client:

```
tar -xzf redis-5.0.8.tar.gz
```

4. Run the following commands to go to the Redis directory and compile the source code of your Redis client:

```
cd redis-5.0.8
```

```
make
```

```
cd src
```

Step 3 Access a DCS instance of a type other than Redis Cluster.

Perform the following procedure to access a DCS Redis 3.0 instance, or a single-node or master/standby DCS Redis 4.0 or 5.0 instance.

```
./redis-cli -h {instance IP} -p 6379 -a {password}
```

NOTE

1. If the instance is password-free, connect it by running the **./redis-cli -h *{instance IP}* -p 6379** command.
2. If the instance is password-protected, connect it by running the **./redis-cli -h *{instance IP}* -p 6379 -a *{password}*** command.

Step 4 Access a DCS instance of the Redis Cluster type.

Perform the following procedure to access a DCS Redis 4.0 or 5.0 instance in Redis Cluster type.

1. Run the following commands to access the chosen DCS Redis instance:

```
./redis-cli -h {dcs_instance_address} -p 6379 -a {password} -c
```

{dcs_instance_address} indicates the IP address of the DCS Redis instance, **6379** is the port used for accessing the instance, *{password}* is the password of the instance, and **-c** is used for accessing Redis Cluster nodes. The IP address and port number are obtained in [Step 1](#).

Example:

```
root@ecs-redis:~/redis-5.0.8/src# ./redis-cli -h 192.168.0.85 -p 6379 -a ***** -c  
192.168.0.85:6379>
```

2. Run the following command to view the Redis Cluster node information:

cluster nodes

Each shard in a Redis Cluster has a master and a replica by default. The proceeding command provides all the information of cluster nodes.

```
192.168.0.85:6379> cluster nodes
0988ae8fd3686074c9afdce73d7878c81a33ddc 192.168.0.231:6379@16379 slave
f0141816260ca5029c56333095f015c7a058f113 0 1568084030
000 3 connected
1a32d809c0b743bd83b5e1c277d5d201d0140b75 192.168.0.85:6379@16379 myself,master - 0
1568084030000 2 connected 5461-10922
c8ad7af9a12cce3c8e416fb67bd6ec9207f0082d 192.168.0.130:6379@16379 slave
1a32d809c0b743bd83b5e1c277d5d201d0140b75 0 1568084031
000 2 connected
7ca218299c254b5da939f8e60a940ac8171adc27 192.168.0.22:6379@16379 master - 0 1568084030000
1 connected 0-5460
f0141816260ca5029c56333095f015c7a058f113 192.168.0.170:6379@16379 master - 0
1568084031992 3 connected 10923-16383
19b1a400815396c6223963b013ec934a657bdc52 192.168.0.161:6379@16379 slave
7ca218299c254b5da939f8e60a940ac8171adc27 0 1568084031
000 1 connected
```

Write operations can only be performed on master nodes. The CRC16 of the key modulo 16384 is taken to compute what is the hash slot of a given key.

As shown in the following, the value of **CRC16 (KEY) mode 16384** determines the hash slot that a given key is located at and redirects the client to the node where the hash slot is located at.

```
192.168.0.170:6379> set hello world
-> Redirected to slot [866] located at 192.168.0.22:6379
OK
192.168.0.22:6379> set happy day
OK
192.168.0.22:6379> set abc 123
-> Redirected to slot [7638] located at 192.168.0.85:6379
OK
192.168.0.85:6379> get hello
-> Redirected to slot [866] located at 192.168.0.22:6379
"world"
192.168.0.22:6379> get abc
-> Redirected to slot [7638] located at 192.168.0.85:6379
"123"
192.168.0.85:6379>
```

----End

Procedure (Windows)

Download the compilation package of the Redis client for Windows. (This is not the source code package.) Decompress the package in any directory, open the CLI tool **cmd.exe**, and go to the directory. Then, run the following command to access the DCS Redis instance:

```
redis-cli.exe -h XXX -p 6379
```

XXX indicates the IP address of the DCS instance and **6379** is an example port number used for accessing a DCS instance. For details about how to obtain the IP address and port number, see [Viewing Details of a DCS Instance](#). Change the IP address and port as required.

3.2.2 Access in Different Languages

3.2.2.1 Java

3.2.2.1.1 Jedis

Access a DCS Redis instance through Jedis on an ECS in the same VPC. For more information on how to use other Redis clients, visit <https://redis.io/clients>.

NOTE

- If a password was set during DCS Redis instance creation, configure the password for connecting to Redis using a Jedis client. Do not hard code the plaintext password.
- When using JedisCluster to connect to a Redis Cluster DCS Redis 4.0 or 5.0 instance, the cluster topology is automatically refreshed. The client needs to reconnect to Redis by itself.

Prerequisites

- The DCS Redis instance you want to access is in the **Running** state.
- An ECS has been created. For more information on how to create ECSs, see the *Elastic Cloud Server User Guide*.
- If the ECS runs the Linux OS, ensure that the Java compilation environment has been installed on the ECS.

Procedure

Step 1 View the IP address/domain name and port number of the DCS Redis instance to be accessed.

For details, see [Viewing Details of a DCS Instance](#).

Step 2 Log in to the ECS where you want to install Docker.

Step 3 Use Maven to add the following dependency to the **pom.xml** file:

```
<dependency>
  <groupId>redis.clients</groupId>
  <artifactId>jedis</artifactId>
  <version>4.1.1</version>
</dependency>
```

Step 4 Access the DCS instance by using Jedis.

Obtain the [source code](#) of the Jedis client. Use either of the following two methods to access a DCS Redis instance through Jedis:

- Single Jedis connection
- Jedis pool

Example code:

1. Example of using Jedis to connect to a single-node, master/standby, or Proxy Cluster DCS Redis instance with a single connection

```
// Creating a connection in password mode
String host = "192.168.0.150";
int port = 6379;
String pwd = "passwd";

Jedis client = new Jedis(host, port);
client.auth(pwd);
client.connect();
```

```
// Run the set command
String result = client.set("key-string", "Hello, Redis!");
System.out.println( String.format("set instruction execution result:%s", result) );
// Run the get command
String value = client.get("key-string");
System.out.println( String.format("get command result:%s", value) );

// Creating a connection in password-free mode
String host = "192.168.0.150";
int port = 6379;

Jedis client = new Jedis(host, port);
client.connect();
// Run the set command
String result = client.set("key-string", "Hello, Redis!");
System.out.println( String.format("set command result:%s", result) );
// Run the get command
String value = client.get("key-string");
System.out.println( String.format("get command result:%s", value) );
```

host indicates the example IP address/domain name of DCS instance and *port* indicates the port number of DCS instance. For details about how to obtain the IP address/domain name and port, see [Step 1](#). Change the IP address and port as required. *pwd* indicates the password used for logging in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation.

2. Example of using Jedis to connect to a single-node, master/standby, or Proxy Cluster DCS Redis instance with connection pooling

```
// Generate configuration information of a Jedis pool
String ip = "192.168.0.150";
int port = 6379;
String pwd = "passwd";
GenericObjectPoolConfig config = new GenericObjectPoolConfig();
config.setTestOnBorrow(false);
config.setTestOnReturn(false);
config.testWhileIdle(true);
config.setMaxTotal(100);
config.setMaxIdle(100);
config.setMaxWaitMillis(2000);
JedisPool pool = new JedisPool(config, ip, port, 100000, pwd);//Generate a Jedis pool when the
application is being initialized
// Get a Jedis connection from the Jedis pool when a service operation occurs
Jedis client = pool.getResource();
try {
    // Run commands
    String result = client.set("key-string", "Hello, Redis!");
    System.out.println( String.format("set command result:%s", result) );
    String value = client.get("key-string");
    System.out.println( String.format("get command result:%s", value) );
} catch (Exception e) {
    // TODO: handle exception
} finally {
    // Return the Jedis connection to the Jedis connection pool after the client's request is processed
    if (null != client) {
        pool.returnResource(client);
    }
} // end of try block
// Destroy the Jedis pool when the application is closed
pool.destroy();

// Configure the connection pool in the password-free mode
String ip = "192.168.0.150";
int port = 6379;
GenericObjectPoolConfig config = new GenericObjectPoolConfig();
config.setTestOnBorrow(false);
config.setTestOnReturn(false);
config.testWhileIdle(true);
```

```
config.setMaxTotal(100);
config.setMaxIdle(100);
config.setMaxWaitMillis(2000);
JedisPool pool = new JedisPool(config, ip, port, 100000); //Generate a JedisPool when the application
is being initialized
// Get a Jedis connection from the Jedis pool when a service operation occurs
Jedis client = pool.getResource();
try {
    // Run commands
    String result = client.set("key-string", "Hello, Redis!");
    System.out.println( String.format("set command result:%s", result) );
    String value = client.get("key-string");
    System.out.println( String.format("get command result:%s", value) );
} catch (Exception e) {
    // TODO: handle exception
} finally {
    // Return the Jedis connection to the Jedis connection pool after the client's request is processed
    if (null != client) {
        pool.returnResource(client);
    }
} // end of try block
// Destroy the Jedis pool when the application is closed
pool.destroy();
```

ip indicates the IP address/domain name of DCS instance and *port* indicates the port number of DCS instance. For details about how to obtain the IP address/domain name and port, see [Step 1](#). Change the IP address and port as required. *pwd* indicates the password used for logging in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation.

Automatic reconnection is supported if the **testOnBorrow** parameter of the connection pool is enabled. When the service tries to obtain a Redis connection from the connection pool, the connection pool checks connections. After detecting a normal connection, the connection pool provides the connection to the service at the cost of performance. If you require high performance, do not enable this parameter and configure the upper-layer application for it to handle exceptions and retries.

3. Example code for connecting to Redis Cluster using a single connection

– With a password

```
//The following shows password-protected access.
int port = 6379;
String host = "192.168.144.37";
//Create JedisCluster.
Set<HostAndPort> nodes = new HashSet<HostAndPort>();
nodes.add(new HostAndPort(host, port));
JedisCluster cluster = new JedisCluster(nodes, 5000, 3000, 10, "password", new
JedisPoolConfig());
cluster.set("key", "value");
System.out.println("Connected to RedisCluster:" + cluster.get("key"));
cluster.close();
```

– Without a password

```
int port = 6379;
String host = "192.168.144.37";
//Create JedisCluster.
Set<HostAndPort> nodes = new HashSet<HostAndPort>();
nodes.add(new HostAndPort(host, port));
JedisCluster cluster = new JedisCluster(nodes);
cluster.set("key", "value");
System.out.println("Connected to RedisCluster:" + cluster.get("key"));
cluster.close();
```

host indicates the example IP address/domain name of DCS instance and *port* indicates the port number of DCS instance. For details about how to obtain the IP address/domain name and port, see [Step 1](#). Change the IP address and port as required. *{password}* indicates the password used to log in to the

chosen DCS Redis instance. This password is defined during DCS Redis instance creation.

Step 5 Compile code according to the **readme** file in the source code of the Jedis client. Run the Jedis client to access the chosen DCS Redis instance.

----End

3.2.2.1.2 Lettuce

Access a Redis Cluster instance through Lettuce on an ECS in the same VPC. For more information on how to use other Redis clients, visit <https://redis.io/clients>.

NOTE

If a password was set during DCS Redis instance creation, configure the password for connecting to Redis using Lettuce. Do not hard code the plaintext password.

To connect to a single-node, master/standby, or Proxy Cluster instance, use the `RedisClient` object of Lettuce. To connect to a Redis Cluster instance, use the `RedisClusterClient` object.

Prerequisites

- The DCS Redis instance you want to access is in the **Running** state.
- An ECS has been created. For more information on how to create ECSs, see the *Elastic Cloud Server User Guide*.
- If the ECS runs the Linux OS, ensure that the Java compilation environment has been installed on the ECS.

Procedure

Step 1 View the IP address/domain name and port number of the DCS Redis instance to be accessed.

For details, see [Viewing Details of a DCS Instance](#).

Step 2 Log in to the ECS.

Step 3 Use Maven to add the following dependency to the **pom.xml** file:

```
<dependency>
  <groupId>io.lettuce</groupId>
  <artifactId>lettuce-core</artifactId>
  <version>6.1.6.RELEASE</version>
</dependency>
```

Step 4 Use Lettuce (a Java client) to connect to the DCS instance.

- Example of using Lettuce to connect to a single-node, master/standby, or Proxy Cluster DCS Redis instance with a single connection

```
// password indicates the connection password. If there is no password, delete "password@". If there is a password and it contains special characters, conversion is required.
RedisClient redisClient = RedisClient.create("redis://password@host:port");
StatefulRedisConnection<String, String> connection = redisClient.connect();
RedisCommands<String, String> syncCommands = connection.sync();
syncCommands.set("key", "value");
System.out.println("Connected to Redis:" + syncCommands.get("key"));
// Close the connection.
connection.close();
// Close the client.
redisClient.shutdown();
```

- Example of using Lettuce to connect to a single-node, master/standby, or Proxy Cluster DCS Redis instance with connection pooling

```
// password indicates the connection password. If there is no password, delete "password@". If there
is a password and it contains special characters, conversion is required.
RedisClient clusterClient = RedisClient.create("redis://password@host:port");
GenericObjectPoolConfig<StatefulRedisConnection<String, String>> genericObjectPoolConfig = new
GenericObjectPoolConfig();
// Connection pool parameters
genericObjectPoolConfig.setMaxIdle(3);
genericObjectPoolConfig.setMinIdle(2);
genericObjectPoolConfig.setMaxTotal(3);
genericObjectPoolConfig.setMaxWaitMillis(-1);
GenericObjectPool<StatefulRedisConnection<String, String>> pool = ConnectionPoolSupport
.createGenericObjectPool(() -> clusterClient.connect(), genericObjectPoolConfig);
// Obtain a connection to perform operations.
try (StatefulRedisConnection<String, String> con = pool.borrowObject()) {
    RedisCommands<String, String> sync = con.sync();
    sync.set("key", "value");
    System.out.println("Connected by pool:" + sync.get("key"));
} catch (Exception e) {
    e.printStackTrace();
}finally {
    // Close the resources.
    pool.close();
    clusterClient.shutdown();
}
```

- Example of using Lettuce to connect to a Redis Cluster

```
// password indicates the connection password. If there is no password, delete "password@". If there
is a password and it contains special characters, conversion is required.
RedisClusterClient redisClient = RedisClusterClient.create("redis://password@host:port");
StatefulRedisClusterConnection<String, String> connection = redisClient.connect();
RedisAdvancedClusterCommands<String, String> syncCommands = connection.sync();
syncCommands.set("key", "value");
System.out.println("Connected to RedisCluster:"+syncCommands.get("key"));
// Close the connection.
connection.close();
// Close the client.
redisClient.shutdown();
```

----End

3.2.2.1.3 Redisson

Access a DCS Redis instance through Redisson on an ECS in the same VPC. For more information about how to use other Redis clients, visit [the Redis official website](#).

NOTE

- If a password was set during DCS Redis instance creation, configure the password for connecting to Redis using Redisson. Do not hard code the plaintext password.
- To connect to a single-node, master/standby, or Proxy Cluster instance, use the **useSingleServer** method of the **SingleServerConfig** object of Redisson. To connect to a Redis Cluster instance, use the **useClusterServers** method of the **ClusterServersConfig** object.

Prerequisites

- A DCS Redis instance has been created and is in the **Running** state.
- An ECS has been created. For details about how to create an ECS, see *Elastic Cloud Server User Guide*.
- If the ECS runs the Linux OS, ensure that the Java compilation environment has been installed on the ECS.

Procedure

Step 1 View the IP address/domain name and port number of the DCS Redis instance to be accessed.

For details, see [Viewing Details of a DCS Instance](#).

Step 2 Log in to the ECS.

Step 3 Use Maven to add the following dependency to the **pom.xml** file:

```
<dependency>
  <groupId>org.redisson</groupId>
  <artifactId>redisson</artifactId>
  <version>3.16.8</version>
</dependency>
```

Step 4 Configure the connection pool.

Recommended keepalive configurations:

```
# ping connection interval. Configuring this parameter will increase Redis load. Set a value based on the
number of connections. The more the connections, the larger the value. Minimum value: 1000. If the
number of active Redis connections exceeds 5000, do not set this parameter.
pingConnectionInterval: 3000
```

The following is a configuration example for a single-node instance. (Set the timeout interval and connection pool size based on the site requirements. The following settings are examples only.)

```
redisson:
  config:
    singleServerConfig:
      # Connection timeout, in milliseconds.
      connectTimeout: 10000
      # Command waiting timeout, in milliseconds.
      timeout: 3000
      # Number of retry times upon a command failure.
      retryAttempts: 3
      # Interval for retrying sending commands, in milliseconds.
      retryInterval: 1500
      # Minimum number of idle connections.
      connectionMinimumIdleSize: 30
      # Connection pool size.
      connectionPoolSize: 50
      # Redis database ID.
      database: 0
      # DNS monitoring interval, in milliseconds.
      dnsMonitoringInterval: 5000
      # ping connection interval.
      pingConnectionInterval: 3000
```

The following is a configuration example for a cluster instance. (Set the timeout interval and connection pool size based on the site requirements.)

```
redisson:
  config:
    clusterServersConfig:
      # Idle connection timeout, in milliseconds.
      idleConnectionTimeout: 100000
      # Connection timeout, in milliseconds.
      connectTimeout: 10000
      # Command waiting timeout, in milliseconds.
      timeout: 3000
      # Number of retry times upon a command failure.
      retryAttempts: 3
```

```
# Interval for retrying sending commands, in milliseconds.
retryInterval: 1500
# Interval for reconnecting a replica node upon a failure.
failedSlaveReconnectionInterval: 3000
# Interval for checking a replica node upon a failure.
failedSlaveCheckInterval: 60000
# Maximum number of subscriptions per connection.
subscriptionsPerConnection: 5
# Client name.
clientName: null
# Minimum number of idle pub/sub connections.
subscriptionConnectionMinimumIdleSize: 1
# Pub/Sub connection pool size.
subscriptionConnectionPoolSize: 50
# Minimum number of idle connections per replica node.
slaveConnectionMinimumIdleSize: 24
# Connection pool size per replica node.
slaveConnectionPoolSize: 64
# Minimum number of idle connections of the master node.
masterConnectionMinimumIdleSize: 24
# Connection pool size of the master node.
masterConnectionPoolSize: 64
# Master node status scan interval, in milliseconds.
scanInterval: 1000
# ping connection interval.
pingConnectionInterval: 3000
# Whether to keep the connection alive.
keepAlive: false
# The tcpNoDelay setting is enabled by default.
tcpNoDelay: false
```

Step 5 Access the DCS instance by using Redisson (a Java client).

- Example of using Redisson to connect to a single-node, master/standby, or Proxy Cluster DCS Redis instance with a single connection

```
Config config = new Config();
SingleServerConfig singleServerConfig = config.useSingleServer();
singleServerConfig.setAddress("redis://host:port");
// singleServerConfig.setPassword("*****");
RedissonClient redisson = Redisson.create(config);
//Test concurrentMap. Data is synchronized to Redis when the put method is used.
ConcurrentMap<String, Object> map = redisson.getMap("FirstMap");
map.put("wanger", "male");
map.put("zhangsan", "nan");
map.put("lisi", "female");
ConcurrentMap resultMap = redisson.getMap("FirstMap");
System.out.println("resultMap==" + resultMap.keySet());
//Test Set
Set mySet = redisson.getSet("MySet");
mySet.add("wanger");
mySet.add("lisi");
Set resultSet = redisson.getSet("MySet");
System.out.println("resultSet===" + resultSet.size());
//Test Queue
Queue myQueue = redisson.getQueue("FirstQueue");
myQueue.add("wanger");
myQueue.add("lili");
myQueue.add("zhangsan");
myQueue.peek();
myQueue.poll();
Queue resultSetQueue = redisson.getQueue("FirstQueue");
System.out.println("resultSetQueue===" + resultSetQueue);
//Close the connection.
redisson.shutdown();
```

- Example of using Redisson to connect to a single-node, master/standby, or Proxy Cluster DCS Redis instance with connection pooling

```
//1. Initialization
Config config = new Config();
SingleServerConfig singleServerConfig = config.useSingleServer();
```

```
singleServerConfig.setAddress("redis://host:6379");
//Set the maximum number of connections in the connection pool of the master node to 500.
singleServerConfig.setConnectionPoolSize(500);
//The connections will be automatically closed and removed from the connection pool. The time unit
is millisecond.
singleServerConfig.setIdleConnectionTimeout(10000);
RedissonClient redisson = Redisson.create(config);
//Test concurrentMap. Data is synchronized to Redis when the put method is used.
ConcurrentMap<String, Object> map = redisson.getMap("FirstMap");
map.put("wanger", "male");
map.put("zhangsan", "nan");
map.put("lisi", "female");
ConcurrentMap resultMap = redisson.getMap("FirstMap");
System.out.println("resultMap==" + resultMap.keySet());
//Test Set
Set mySet = redisson.getSet("MySet");
mySet.add("wanger");
mySet.add("lisi");
Set resultSet = redisson.getSet("MySet");
System.out.println("resultSet===" + resultSet.size());
//Test Queue
Queue myQueue = redisson.getQueue("FirstQueue");
myQueue.add("wanger");
myQueue.add("lili");
myQueue.add("zhangsan");
myQueue.peek();
myQueue.poll();
Queue resultQueue = redisson.getQueue("FirstQueue");
System.out.println("resultQueue===" + resultQueue);
//Close the connection.
redisson.shutdown();
```

- Example of using Redisson to connect to a Redis Cluster

```
Config config = new Config();
ClusterServersConfig clusterServersConfig = config.useClusterServers();
clusterServersConfig.addNodeAddress("redis://host:port");
//Set a password.
// clusterServersConfig.setPassword("*****");
RedissonClient redisson = Redisson.create(config);
ConcurrentMap<String, Object> map = redisson.getMap("FirstMap");
map.put("wanger", "male");
map.put("zhangsan", "nan");
map.put("lisi", "female");
ConcurrentMap resultMap = redisson.getMap("FirstMap");
System.out.println("resultMap==" + resultMap.keySet());
//2. Test Set
Set mySet = redisson.getSet("MySet");
mySet.add("wanger");
mySet.add("lisi");
Set resultSet = redisson.getSet("MySet");
System.out.println("resultSet===" + resultSet.size());
//3. Test Queue
Queue myQueue = redisson.getQueue("FirstQueue");
myQueue.add("wanger");
myQueue.add("lili");
myQueue.add("zhangsan");
myQueue.peek();
myQueue.poll();
Queue resultQueue = redisson.getQueue("FirstQueue");
System.out.println("resultQueue===" + resultQueue);
//Close the connection.
redisson.shutdown();
```

----End

3.2.2.2 Lettuce Integration with Spring Boot

Prerequisites

- A DCS Redis instance has been created and is in the **Running** state.
- An ECS has been created. For details about how to create an ECS, see *Elastic Cloud Server User Guide*.
- If the ECS runs the Linux OS, ensure that the Java compilation environment has been installed on the ECS.

Procedure

Step 1 View the IP address/domain name and port number of the DCS Redis instance to be accessed.

For details, see [Viewing Details of a DCS Instance](#).

Step 2 Log in to the ECS.

Step 3 Use Maven to add the following dependency to the **pom.xml** file:

NOTE

- Since Spring Boot 2.0, Lettuce is used as the default client for connections.
- Spring Boot 2.6.6 and Lettuce 6.1.8 are used.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

Step 4 Use Spring Boot integrated with Lettuce to connect to the instance.

- Example of using Spring Boot and Lettuce to connect to a single-node, master/standby, or Proxy Cluster DCS Redis instance with a single connection
 - a. Add the Redis configuration to the **application.properties** configuration file.

```
spring.redis.host=host
spring.redis.database=0
spring.redis.password=pwd
spring.redis.port=port
```

- b. Redis configuration class RedisConfiguration

```
@Bean
public RedisTemplate<String, Object> redisTemplate(LettuceConnectionFactory
lettuceConnectionFactory) {
    RedisTemplate<String, Object> template = new RedisTemplate<>();
    template.setConnectionFactory(lettuceConnectionFactory);
    // Replace the default JdkSerializationRedisSerializer with Jackson2JsonRedisSerializer to
    serialize and deserialize the Redis value.
    Jackson2JsonRedisSerializer<Object> jackson2JsonRedisSerializer = new
Jackson2JsonRedisSerializer<>(Object.class);
    ObjectMapper mapper = new ObjectMapper();
    mapper.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);
    mapper.activateDefaultTyping(LaissezFaireSubTypeValidator.instance,
    ObjectMapper.DefaultTyping.NON_FINAL, JsonTypeInfo.As.PROPERTY);
    jackson2JsonRedisSerializer.setObjectMapper(mapper);
    StringRedisSerializer stringRedisSerializer = new StringRedisSerializer();
    // String serialization of keys
```

```
template.setKeySerializer(stringRedisSerializer);
// String serialization of hash keys
template.setHashKeySerializer(stringRedisSerializer);
// Jackson serialization of values
template.setValueSerializer(jackson2JsonRedisSerializer);
// Jackson serialization of hash values
template.setHashValueSerializer(jackson2JsonRedisSerializer);
template.afterPropertiesSet();
return template;
}
```

c. Redis operation class RedisUtil

```
/**
 * Obtain data from the cache.
 * @param key
 * @return value
 */
public Object get(String key){
    return key==null?null:redisTemplate.opsForValue().get(key);
}

/**
 * Write data to the cache.
 * @param key
 * @param value
 * @return true (successful) false (failed)
 */
public boolean set(String key,Object value) {
    try {
        redisTemplate.opsForValue().set(key, value);
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}
```

d. Write the controller class for testing.

```
@RestController
public class HelloRedis {
    @Autowired
    RedisUtil redisUtil;

    @RequestMapping("/setParams")
    @ResponseBody
    public String setParams(String name) {
        redisUtil.set("name", name);
        return "success";
    }

    @RequestMapping("/getParams")
    @ResponseBody
    public String getParams(String name) {
        System.out.println("-----" + name + "-----");
        String retName = redisUtil.get(name) + "";
        return retName;
    }
}
```

- Example of using Spring Boot and Lettuce to connect to a single-node, master/standby, or Proxy Cluster DCS Redis instance with connection pooling

a. Add the following dependency in addition to the preceding Maven dependency:

```
<dependency>
<groupId>org.apache.commons</groupId>
<artifactId>commons-pool2</artifactId>
</dependency>
```

- b. Add the Redis configuration to the **application.properties** configuration file.

```
spring.redis.host=host
spring.redis.database=0
spring.redis.password=pwd
spring.redis.port=port
# Connection timeout.
spring.redis.timeout=1000
# Maximum number of connections in the connection pool. A negative value indicates no limit.
spring.redis.lettuce.pool.max-active=50
# Minimum number of idle connections in the connection pool.
spring.redis.lettuce.pool.min-idle=5
# Maximum number of idle connections in the connection pool.
spring.redis.lettuce.pool.max-idle=50
# Maximum time for waiting for connections in the connection pool. A negative value indicates no limit.
spring.redis.lettuce.pool.max-wait=5000
# Interval for scheduling an eviction thread.
spring.redis.pool.time-between-eviction-runs-millis=2000
```

- c. Redis connection configuration class RedisConfiguration

```
@Bean
public RedisTemplate<String, Object> redisTemplate(LettuceConnectionFactory
lettuceConnectionFactory) {
    lettuceConnectionFactory.setShareNativeConnection(false);
    RedisTemplate<String, Object> template = new RedisTemplate<>();
    template.setConnectionFactory(lettuceConnectionFactory);
    // Use Jackson2JsonRedisSerializer to replace the default JdkSerializationRedisSerializer to
    serialize and deserialize the Redis value.
    Jackson2JsonRedisSerializer<Object> jackson2JsonRedisSerializer = new
Jackson2JsonRedisSerializer<>(Object.class);
    ObjectMapper mapper = new ObjectMapper();
    mapper.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);
    mapper.activateDefaultTyping(LaissezFaireSubTypeValidator.instance,
ObjectMapper.DefaultTyping.NON_FINAL, JsonTypeInfo.As.PROPERTY);
    jackson2JsonRedisSerializer.setObjectMapper(mapper);
    StringRedisSerializer stringRedisSerializer = new StringRedisSerializer();
    // String serialization of keys
    template.setKeySerializer(stringRedisSerializer);
    // String serialization of hash keys
    template.setHashKeySerializer(stringRedisSerializer);
    // Jackson serialization of values
    template.setValueSerializer(jackson2JsonRedisSerializer);
    // Jackson serialization of hash values
    template.setHashValueSerializer(jackson2JsonRedisSerializer);
    template.afterPropertiesSet();
    return template;
}
```

- Example code for using Spring Boot and Lettuce to connect to Redis Cluster using a single connection

- a. Add the Redis configuration to the **application.properties** configuration file.

```
spring.redis.cluster.nodes=host:port
spring.redis.cluster.max-redirects=3
spring.redis.password= pwd
# Automated refresh interval
spring.redis.lettuce.cluster.refresh.period=60
# Enable automated refresh
spring.redis.lettuce.cluster.refresh.adaptive=true
spring.redis.timeout=60
```

- b. Redis configuration class RedisConfiguration (automated topology refresh must be enabled).

```
@Bean
public LettuceConnectionFactory lettuceConnectionFactory() {
    String[] nodes = clusterNodes.split(",");
    List<RedisNode> listNodes = new ArrayList();
```

```
for (String node : nodes) {
    String[] ipAndPort = node.split(":");
    RedisNode redisNode = new RedisNode(ipAndPort[0], Integer.parseInt(ipAndPort[1]));
    listNodes.add(redisNode);
}
RedisClusterConfiguration redisClusterConfiguration = new RedisClusterConfiguration();
redisClusterConfiguration.setClusterNodes(listNodes);
redisClusterConfiguration.setPassword(password);
redisClusterConfiguration.setMaxRedirects(maxRedirects);
// Configure automated topology refresh.
ClusterTopologyRefreshOptions topologyRefreshOptions =
ClusterTopologyRefreshOptions.builder()
    .enablePeriodicRefresh(Duration.ofSeconds(period)) // Refresh the topology periodically.
    .enableAllAdaptiveRefreshTriggers() // Refresh the topology based on events.
    .build();

ClusterClientOptions clusterClientOptions = ClusterClientOptions.builder()
    // Redis command execution timeout. Only when the command execution times out will a
    // reconnection be triggered using the new topology.
    .timeoutOptions(TimeoutOptions.enabled(Duration.ofSeconds(period)))
    .topologyRefreshOptions(topologyRefreshOptions)
    .build();
LettuceClientConfiguration clientConfig = LettucePoolingClientConfiguration.builder()
    .commandTimeout(Duration.ofSeconds(timeout))
    .readFrom(ReadFrom.REPLICA_PREFERRED) // Preferentially read data from the replicas.
    .clientOptions(clusterClientOptions)
    .build();
LettuceConnectionFactory factory = new
LettuceConnectionFactory(redisClusterConfiguration, clientConfig);
return factory;
}

@Bean
public RedisTemplate<String, Object> redisTemplate(LettuceConnectionFactory
lettuceConnectionFactory) {
    RedisTemplate<String, Object> template = new RedisTemplate<>();
    template.setConnectionFactory(lettuceConnectionFactory);
    // Use Jackson2JsonRedisSerializer to replace the default JdkSerializationRedisSerializer to
    // serialize and deserialize the Redis value.
    Jackson2JsonRedisSerializer<Object> jackson2JsonRedisSerializer = new
Jackson2JsonRedisSerializer<>(Object.class);
    ObjectMapper mapper = new ObjectMapper();
    mapper.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);
    mapper.activateDefaultTyping(LaissezFaireSubTypeValidator.instance,
        ObjectMapper.DefaultTyping.NON_FINAL, JsonTypeInfo.As.PROPERTY);
    jackson2JsonRedisSerializer.setObjectMapper(mapper);
    StringRedisSerializer stringRedisSerializer = new StringRedisSerializer();
    // String serialization of keys
    template.setKeySerializer(stringRedisSerializer);
    // String serialization of hash keys
    template.setHashKeySerializer(stringRedisSerializer);
    // Jackson serialization of values
    template.setValueSerializer(jackson2JsonRedisSerializer);
    // Jackson serialization of hash values
    template.setHashValueSerializer(jackson2JsonRedisSerializer);
    template.afterPropertiesSet();
    return template;
}
```

- Example code for using Spring Boot and Lettuce to connect to Redis Cluster with connection pooling
 - a. Add the Redis configuration to the **application.properties** configuration file.

```
spring.redis.cluster.nodes=host:port
spring.redis.cluster.max-redirects=3
spring.redis.password=pwd
spring.redis.lettuce.cluster.refresh.period=60
spring.redis.lettuce.cluster.refresh.adaptive=true
# Connection timeout.
```

```

spring.redis.timeout=60s
# Maximum number of connections in the connection pool. A negative value indicates no limit.
spring.redis.lettuce.pool.max-active=50
# Minimum number of idle connections in the connection pool.
spring.redis.lettuce.pool.min-idle=5
# Maximum number of idle connections in the connection pool.
spring.redis.lettuce.pool.max-idle=50
# Maximum time for waiting for connections in the connection pool. A negative value indicates
no limit.
spring.redis.lettuce.pool.max-wait=5000
# Interval for scheduling an eviction thread.
spring.redis.lettuce.pool.time-between-eviction-runs=2000

```

b. Redis configuration class `RedisConfiguration` (automated topology refresh must be enabled).

```

@Bean
public LettuceConnectionFactory lettuceConnectionFactory() {
    GenericObjectPoolConfig genericObjectPoolConfig = new GenericObjectPoolConfig();
    genericObjectPoolConfig.setMaxIdle(maxIdle);
    genericObjectPoolConfig.setMinIdle(minIdle);
    genericObjectPoolConfig.setMaxTotal(maxActive);
    genericObjectPoolConfig.setMaxWait(Duration.ofMillis(maxWait));

    genericObjectPoolConfig.setTimeBetweenEvictionRuns(Duration.ofMillis(timeBetweenEvictionRunsMillis));
    String[] nodes = clusterNodes.split(",");
    List<RedisNode> listNodes = new ArrayList();
    for (String node : nodes) {
        String[] ipAndPort = node.split(":");
        RedisNode redisNode = new RedisNode(ipAndPort[0], Integer.parseInt(ipAndPort[1]));
        listNodes.add(redisNode);
    }
    RedisClusterConfiguration redisClusterConfiguration = new RedisClusterConfiguration();
    redisClusterConfiguration.setClusterNodes(listNodes);
    redisClusterConfiguration.setPassword(password);
    redisClusterConfiguration.setMaxRedirects(maxRedirects);
    // Configure automated topology refresh.
    ClusterTopologyRefreshOptions topologyRefreshOptions =
ClusterTopologyRefreshOptions.builder()
    .enablePeriodicRefresh(Duration.ofSeconds(period)) // Refresh the topology periodically.
    .enableAllAdaptiveRefreshTriggers() // Refresh the topology based on events.
    .build();

    ClusterClientOptions clusterClientOptions = ClusterClientOptions.builder()
    // Redis command execution timeout. Only when the command execution times out will a
reconnection be triggered using the new topology.
    .timeoutOptions(TimeoutOptions.enabled(Duration.ofSeconds(period)))
    .topologyRefreshOptions(topologyRefreshOptions)
    .build();
    LettuceClientConfiguration clientConfig = LettucePoolingClientConfiguration.builder()
    .commandTimeout(Duration.ofSeconds(timeout))
    .poolConfig(genericObjectPoolConfig)
    .readFrom(ReadFrom.REPLICA_PREFERRED) // Preferentially read data from the replicas.
    .clientOptions(clusterClientOptions)
    .build();
    LettuceConnectionFactory factory = new
LettuceConnectionFactory(redisClusterConfiguration, clientConfig);
    return factory;
}

@Bean
public RedisTemplate<String, Object> redisTemplate(LettuceConnectionFactory
lettuceConnectionFactory) {
    lettuceConnectionFactory.setShareNativeConnection(false);
    RedisTemplate<String, Object> template = new RedisTemplate<>();
    template.setConnectionFactory(lettuceConnectionFactory);
    // Use Jackson2JsonRedisSerializer to replace the default JdkSerializationRedisSerializer to
serialize and deserialize the Redis value.
    Jackson2JsonRedisSerializer<Object> jackson2JsonRedisSerializer = new
Jackson2JsonRedisSerializer<>(Object.class);

```



```
    ObjectMapper mapper = new ObjectMapper();
    mapper.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);
    mapper.activateDefaultTyping(LaissezFaireSubTypeValidator.instance,
        ObjectMapper.DefaultTyping.NON_FINAL, JsonTypeInfo.As.PROPERTY);
    Jackson2JsonRedisSerializer.setObjectMapper(mapper);
    StringRedisSerializer stringRedisSerializer = new StringRedisSerializer();
    // String serialization of keys
    template.setKeySerializer(stringRedisSerializer);
    // String serialization of hash keys
    template.setHashKeySerializer(stringRedisSerializer);
    // Jackson serialization of values
    template.setValueSerializer(jackson2JsonRedisSerializer);
    // Jackson serialization of hash values
    template.setHashValueSerializer(jackson2JsonRedisSerializer);
    template.afterPropertiesSet();
    return template;
}
```

host is the IP address/domain name of the DCS instance, **port** is the port number of the DCS instance, and **pwd** is the password of the DCS instance. Specify these parameters as required before running the code. Connection pooling is recommended. Adjust parameters such as **TimeOut**, **MaxTotal** (maximum number of connections), **MinIdle** (minimum number of idle connections), **MaxIdle** (maximum number of idle connections), and **MaxWait** (maximum waiting time) based on service requirements.

----End

3.2.2.3 Clients in Python

Access a DCS Redis instance through redis-py on an ECS in the same VPC. For more information about how to use other Redis clients, visit [the Redis official website](#).

NOTE

Use redis-py to connect to single-node, master/standby, and Proxy Cluster instances and redis-py-cluster to connect to Redis Cluster instances.

Prerequisites

- A DCS Redis instance has been created and is in the **Running** state.
- An ECS has been created. For details about how to create an ECS, see *Elastic Cloud Server User Guide*.
- If the ECS runs the Linux OS, ensure that the Python compilation environment has been installed on the ECS.

Procedure

Step 1 View the IP address/domain name and port number of the DCS Redis instance to be accessed.

For details, see [Viewing Details of a DCS Instance](#).

Step 2 Log in to the ECS.

The following uses CentOS as an example to describe how to access an instance using a Python client.

Step 3 Access the DCS Redis instance.

If the system does not provide Python, run the following **yum** command to install it:

yum install python

NOTE

The Python version must be 3.6 or later. If the default Python version is earlier than 3.6, perform the following operations to change it:

1. Run the **rm -rf python** command to delete the Python symbolic link.
 2. Run the **ln -s python.X.X.X python** command to create another Python link. In the command, *X.X.X* indicates the Python version number.
- If the instance is a single-node, master/standby, or Proxy Cluster instance:
 - a. Install Python and redis-py.
 - i. If the system does not provide Python, run the following **yum** command to install it.
 - ii. Run the following command to download and decompress the redis-py package:

```
wget https://github.com/andymccurdy/redis-py/archive/master.zip
```

```
unzip master.zip
```

- iii. Go to the directory where the decompressed redis-py package is saved, and install redis-py.

python setup.py install

After the installation, run the **python** command. redis-py have been successfully installed if the following command output is displayed:

Figure 3-1 Running the python command

```
[root@ecs-~]# python
Python 3.6.8 (default, Nov 16 2020, 16:55:22)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import redis
>>>
```

- b. Use the redis-py client to connect to the instance. In the following steps, commands are executed in CLI mode. (Alternatively, write the commands into a Python script and then execute the script.)
 - i. Run the **python** command to enter the CLI mode. You have entered CLI mode if the following command output is displayed:

Figure 3-2 Entering the CLI mode

```
[root@ecs-~]# python
Python 3.6.8 (default, Nov 16 2020, 16:55:22)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import redis
>>>
```

- ii. Run the following command to access the chosen DCS Redis instance:

```
r = redis.StrictRedis(host='XXX.XXX.XXX.XXX', port=6379, password='*****');
```

`XXX.XXX.XXX.XXX` indicates the IP address/domain name of the DCS instance and **6379** is an example port number of the instance. For details about how to obtain the IP address/domain name and port, see [Step 1](#). Change the IP address/domain name and port as required. `*****` indicates the password used for logging in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation.

You have successfully accessed the instance if the following command output is displayed. Enter commands to perform read and write operations on the database.

Figure 3-3 Redis connected successfully

```
>>> r = redis.StrictRedis(host='192.168.0.9', port=6379, password='*****');
>>> r.set("foo", "bar")
True
>>> print(r.get("foo"))
b'bar'
>>> _
```

- If the instance is a Redis Cluster instance:
 - a. Install the `redis-py-cluster` client.
 - i. Download the released version.
wget https://github.com/Grokzen/redis-py-cluster/releases/download/2.1.3/redis-py-cluster-2.1.3.tar.gz
 - ii. Decompress the package.
tar -xvf redis-py-cluster-2.1.3.tar.gz
 - iii. Go to the directory where the decompressed `redis-py-cluster` package is saved, and install `redis-py-cluster`.
python setup.py install
 - b. Access the DCS Redis instance by using `redis-py-cluster`.
In the following steps, commands are executed in CLI mode.
(Alternatively, write the commands into a Python script and then execute the script.)
 - i. Run the **python** command to enter the CLI mode.
 - ii. Run the following command to access the chosen DCS Redis instance:

```
>>> from rediscluster import RedisCluster
>>> startup_nodes = [{"host": "192.168.0.143", "port": "6379"}]
>>> rc = RedisCluster(startup_nodes=startup_nodes, decode_responses=True)
>>> rc.set("foo", "bar")
True
>>> print(rc.get("foo"))
'bar'
```

----End

3.2.2.4 go-redis

Access a DCS Redis instance through go-redis on an ECS in the same VPC. For more information about how to use other Redis clients, visit [the Redis official website](#).

Prerequisites

- A DCS Redis instance has been created and is in the **Running** state.
- An ECS has been created. For details about how to create an ECS, see *Elastic Cloud Server User Guide*.

Procedure

Step 1 View the IP address/domain name and port number of the DCS Redis instance to be accessed.

For details, see [Viewing Details of a DCS Instance](#).

Step 2 Log in to the ECS.

A Windows ECS is used as an example.

Step 3 Install Visual Studio Community 2017 on the ECS.

Step 4 Start Visual Studio and create a project. The project name can be customized. In this example, the project name is set to **redisdemo**.

Step 5 Import the dependency package of go-redis and enter **go get github.com/go-redis/redis** on the terminal.

Step 6 Write the following code:

```
package main

import (
    "fmt"
    "github.com/go-redis/redis"
)

func main() {
    // Single-node
    rdb := redis.NewClient(&redis.Options{
        Addr:     "host:port",
        Password: "*****", // no password set
        DB:      0, // use default DB
    })

    val, err := rdb.Get("key").Result()
    if err != nil {
        if err == redis.Nil {
            fmt.Println("key does not exists")
            return
        }
        panic(err)
    }
    fmt.Println(val)

    //Cluster
    rdbCluster := redis.NewClusterClient(&redis.ClusterOptions{
        Addrs: []string{"host:port"},
        Password: "*****",
    })
    val1, err1 := rdbCluster.Get("key").Result()
```

```
if err1 != nil {
    if err == redis.Nil {
        fmt.Println("key does not exists")
        return
    }
    panic(err)
}
fmt.Println(val1)
}
```

host:port are the IP address/domain name and port number of the DCS Redis instance. For details about how to obtain the IP address/domain name and port, see [Step 1](#). Change the IP address/domain name and port as required. ********* indicates the password used to log in to the DCS Redis instance. This password is defined during DCS Redis instance creation.

Step 7 Run the `go build -o test main.go` command to package the code into an executable file, for example, `test`.

CAUTION

To run the package in the Linux OS, set the following parameters before packaging:

set GOARCH=amd64

set GOOS=linux

Step 8 Run the `./test` command to access the DCS instance.

----End

3.2.2.5 hiredis in C++

Access a DCS Redis instance through hiredis on an ECS in the same VPC. For more information about how to use other Redis clients, visit [the Redis official website](#).

NOTE

The operations described in this section apply only to single-node, master/standby, and Proxy Cluster instances. To use C++ to connect to a Redis Cluster instance, see the [C++ Redis client description](#).

Prerequisites

- A DCS Redis instance has been created and is in the **Running** state.
- An ECS has been created. For details about how to create an ECS, see *Elastic Cloud Server User Guide*.
- If the ECS runs the Linux OS, ensure that the GCC compilation environment has been installed on the ECS.

Procedure

Step 1 View the IP address/domain name and port number of the DCS Redis instance to be accessed.

For details, see [Viewing Details of a DCS Instance](#).

Step 2 Log in to the ECS.

The following uses CentOS as an example to describe how to access an instance in C++.

Step 3 Install GCC, Make, and hiredis.

If the system does not provide a compiling environment, run the following **yum** command to install the environment:

```
yum install gcc make
```

Step 4 Run the following command to download and decompress the hiredis package:

```
wget https://github.com/redis/hiredis/archive/master.zip
```

```
unzip master.zip
```

Step 5 Go to the directory where the decompressed hiredis package is saved, and compile and install hiredis.

```
make
```

```
make install
```

Step 6 Access the DCS instance by using hiredis.

The following describes connection and password authentication of hiredis. For more information on how to use hiredis, visit the Redis official website.

1. Edit the sample code for connecting to a DCS instance, and then save the code and exit.

vim connRedis.c

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <hiredis.h>
int main(int argc, char **argv) {
    unsigned int j;
    redisContext *conn;
    redisReply *reply;
    if (argc < 3) {
        printf("Usage: example {instance_ip_address} 6379 {password}\n");
        exit(0);
    }
    const char *hostname = argv[1];
    const int port = atoi(argv[2]);
    const char *password = argv[3];
    struct timeval timeout = { 1, 500000 }; // 1.5 seconds
    conn = redisConnectWithTimeout(hostname, port, timeout);
    if (conn == NULL || conn->err) {
        if (conn) {
            printf("Connection error: %s\n", conn->errstr);
            redisFree(conn);
        } else {
            printf("Connection error: can't allocate redis context\n");
        }
    }
    exit(1);
}
/* AUTH */
reply = redisCommand(conn, "AUTH %s", password);
printf("AUTH: %s\n", reply->str);
freeReplyObject(reply);
```

```
/* Set */
reply = redisCommand(conn,"SET %s %s", "welcome", "Hello, DCS for Redis!");
printf("SET: %s\n", reply->str);
freeReplyObject(reply);

/* Get */
reply = redisCommand(conn,"GET welcome");
printf("GET welcome: %s\n", reply->str);
freeReplyObject(reply);

/* Disconnects and frees the context */
redisFree(conn);
return 0;
}
```

2. Run the following command to compile the code:

```
gcc connRedis.c -o connRedis -I /usr/local/include/hiredis -lhiredis
```

If an error is reported, locate the directory where the **hiredis.h** file is saved and modify the compilation command.

After the compilation, an executable **connRedis** file is obtained.

3. Run the following command to access the chosen DCS Redis instance:

```
./connRedis {redis_ip_address} 6379 {password}
```

{redis_instance_address} indicates the IP address/domain name of DCS instance and **6379** is an example port number of DCS instance. For details about how to obtain the IP address/domain name and port, see [Step 1](#). Change the IP address/domain name and port as required. *{password}* indicates the password used to log in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation.

You have successfully accessed the instance if the following command output is displayed:

```
AUTH: OK
SET: OK
GET welcome: Hello, DCS for Redis!
```

NOTICE

If an error is reported, indicating that the hiredis library files cannot be found, run the following commands to copy related files to the system directories and add dynamic links:

```
mkdir /usr/lib/hiredis
```

```
cp /usr/local/lib/libhiredis.so.0.13 /usr/lib/hiredis/
```

```
mkdir /usr/include/hiredis
```

```
cp /usr/local/include/hiredis/hiredis.h /usr/include/hiredis/
```

```
echo '/usr/local/lib' >>> /etc/ld.so.conf
```

```
ldconfig
```

Replace the locations of the **so** and **.h** files with actual ones before running the commands.

----End

3.2.2.6 C#

Access a DCS Redis instance through C# Client StackExchange.Redis on an ECS in the same VPC. For more information about how to use other Redis clients, visit [the Redis official website](#).

Prerequisites

- A DCS Redis instance has been created and is in the **Running** state.
- An ECS has been created. For details about how to create an ECS, see *Elastic Cloud Server User Guide*.
- If the ECS runs the Linux OS, ensure that the GCC compilation environment has been installed on the ECS.

Procedure

Step 1 View the IP address/domain name and port number of the DCS Redis instance to be accessed.

For details, see [Viewing Details of a DCS Instance](#).

Step 2 Log in to the ECS.

A Windows ECS is used as an example.

Step 3 Install Visual Studio Community 2017 on the ECS.

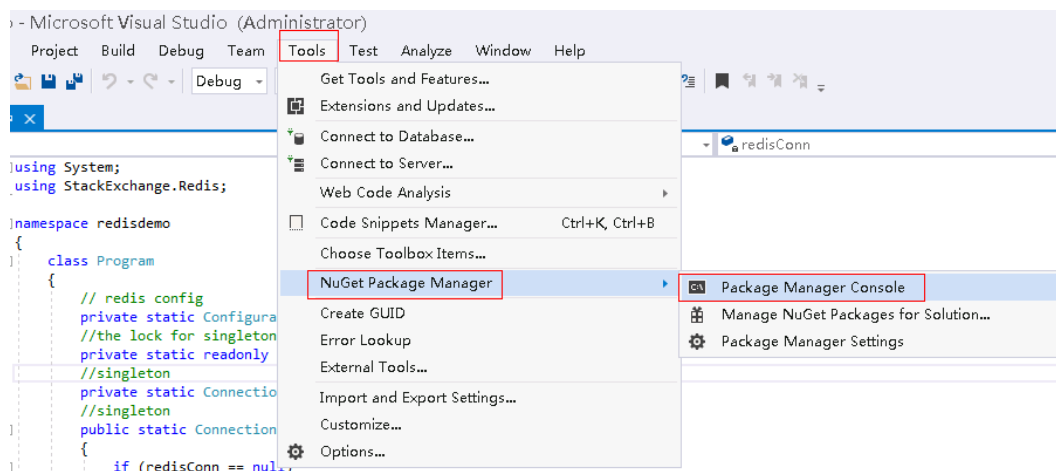
Step 4 Start Visual Studio 2017 and create a project.

Set the project name to **redisdemo**.

Step 5 Install StackExchange.Redis by using the NuGet package manager of Visual Studio.

Access the NuGet package manager console according to [Figure 3-4](#), and enter **Install-Package StackExchange.Redis - Version 2.2.79**. (The version number is optional).

Figure 3-4 Accessing the NuGet package manager console



Step 6 Write the following code, and use the String Set and Get methods to test the connection.


```
using System;
using StackExchange.Redis;

namespace reisdemo
{
    class Program
    {
        // redis config
        private static ConfigurationOptions connDCS =
        ConfigurationOptions.Parse("10.10.38.233:6379,password=*****,connectTimeout=2000");
        //the lock for singleton
        private static readonly object Locker = new object();
        //singleton
        private static ConnectionMultiplexer redisConn;
        //singleton
        public static ConnectionMultiplexer getRedisConn()
        {
            if (redisConn == null)
            {
                lock (Locker)
                {
                    if (redisConn == null || !redisConn.IsConnected)
                    {
                        redisConn = ConnectionMultiplexer.Connect(connDCS);
                    }
                }
            }
            return redisConn;
        }
        static void Main(string[] args)
        {
            redisConn = getRedisConn();
            var db = redisConn.GetDatabase();
            //set get
            string strKey = "Hello";
            string strValue = "DCS for Redis!";
            Console.WriteLine( strKey + ", " + db.StringGet(strKey));

            Console.ReadLine();
        }
    }
}
```

`10.10.38.233:6379` contains an example IP address/domain name and port number of the DCS Redis instance. For details about how to obtain the IP address/domain name and port, see [Step 1](#). Change the IP address/domain name and port as required. `*****` indicates the password used for logging in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation.

Step 7 Run the code. You have successfully accessed the instance if the following command output is displayed:

```
Hello, DCS for Redis!
```

For more information about other commands of StackExchange.Redis, visit [StackExchange.Redis](#).

----End

3.2.2.7 PHP

3.2.2.7.1 phpredis

Access a DCS Redis instance through phpredis on an ECS in the same VPC. For more information about how to use other Redis clients, visit [the Redis official website](#).

NOTE

The operations described in this section apply only to single-node, master/standby, and Proxy Cluster instances. To use phpredis to connect to a Redis Cluster instance, see the [phpredis description](#).

Prerequisites

- A DCS Redis instance has been created and is in the **Running** state.
- An ECS has been created. For details about how to create an ECS, see *Elastic Cloud Server User Guide*.
- If the ECS runs the Linux OS, ensure that the GCC compilation environment has been installed on the ECS.

Procedure

Step 1 View the IP address/domain name and port number of the DCS Redis instance to be accessed.

For details, see [Viewing Details of a DCS Instance](#).

Step 2 Log in to the ECS.

The following uses CentOS as an example to describe how to access an instance through phpredis.

Step 3 Install GCC-C++ and Make compilation components.

```
yum install gcc-c++ make
```

Step 4 Install the PHP development package and CLI tool.

Run the following **yum** command to install the PHP development package:

```
yum install php-devel php-common php-cli
```

After the installation is complete, run the following command to query the PHP version and check whether the installation is successful:

```
php --version
```

Step 5 Install the phpredis client.

1. Download the source phpredis package.

```
wget http://pecl.php.net/get/redis-5.3.7.tgz
```

This version is used as an example. To download phpredis clients of other versions, visit the Redis or PHP official website.

2. Decompress the source phpredis package.

```
tar -zxvf redis-5.3.7.tgz  
cd redis-5.3.7
```

3. Command before compilation.

phpize

4. Configure the **php-config** file.

./configure --with-php-config=/usr/bin/php-config

The location of the file varies depending on the OS and PHP installation mode. You are advised to locate the directory where the file is saved before the configuration.

find / -name php-config

5. Compile and install the phpredis client.

make && make install

6. After the installation, add the **extension** configuration in the **php.ini** file to reference the Redis module.

vim /etc/php.ini

Add the following configuration:

```
extension = "/usr/lib64/php/modules/redis.so"
```

 **NOTE**

The **redis.so** file may be saved in a different directory from **php.ini**. Run the following command to locate the directory:

find / -name php.ini

7. Save the configuration and exit. Then, run the following command to check whether the extension takes effect:

php -m |grep redis

If the command output contains **redis**, the phpredis client environment has been set up.

Step 6 Access the DCS instance by using phpredis.

1. Edit a **redis.php** file.

```
<?php
$redis_host = "{redis_instance_address}";
$redis_port = 6379;
$user_pwd = "{password}";
$redis = new Redis();
if ($redis->connect($redis_host, $redis_port) == false) {
    die($redis->getLastError());
}
if ($redis->auth($user_pwd) == false) {
    die($redis->getLastError());
}
if ($redis->set("welcome", "Hello, DCS for Redis!") == false) {
    die($redis->getLastError());
}
$value = $redis->get("welcome");
echo $value;
$redis->close();
?>
```

{redis_instance_address} indicates the IP address/domain name of DCS instance and *6379* is an example port number of DCS instance. For details about how to obtain the IP address/domain name and port, see [Step 1](#). Change the IP address/domain name and port as required. *{password}* indicates the password used to log in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation. If password-free access is enabled, shield the **if** statement for password authentication.

2. Run the **php redis.php** command to access the DCS instance.

----End

3.2.2.7.2 Predis

Access a DCS Redis instance through Predis on an ECS in the same VPC. For more information about how to use other Redis clients, visit [the Redis official website](#).

Prerequisites

- A DCS Redis instance has been created and is in the **Running** state.
- An ECS has been created. For details about how to create an ECS, see *Elastic Cloud Server User Guide*.
- If the ECS runs the Linux OS, ensure that the PHP compilation environment has been installed on the ECS.

Procedure

- Step 1** View the IP address/domain name and port number of the DCS Redis instance to be accessed.

For details, see [Viewing Details of a DCS Instance](#).

- Step 2** Log in to the ECS.

- Step 3** Install the PHP development package and CLI tool. Run the following **yum** command:

```
yum install php-devel php-common php-cli
```

- Step 4** After the installation is complete, check the version number to ensure that the installation is successful.

```
php --version
```

- Step 5** Download the Predis package to the **/usr/share/php** directory.

1. Run the following command to download the Predis source file:

```
wget https://github.com/predis/predis/archive/refs/tags/v1.1.10.tar.gz
```

 **NOTE**

This version is used as an example. To download Predis clients of other versions, visit the Redis or PHP official website.

2. Run the following commands to decompress the source Predis package:

```
tar -zxvf predis-1.1.10.tar.gz
```

3. Rename the decompressed Predis directory **predis** and move it to **/usr/share/php/**.

```
mv predis-1.1.10 predis
```

- Step 6** Edit a file used to connect to Redis.

- Example of using **redis.php** to connect to a single-node, master/standby, or Proxy Cluster DCS Redis instance:

```
<?php  
require 'predis/autoload.php';
```

```
Predis\Autoloader::register();
$client = new Predis\Client([
    'scheme' => 'tcp' ,
    'host'   => '{redis_instance_address}' ,
    'port'   => {port} ,
    'password' => '{password}'
]);
$client->set('foo', 'bar');
$value = $client->get('foo');
echo $value;
?>
```

- Example code for using **redis-cluster.php** to connect to Redis Cluster:

```
<?php
require 'predis/autoload.php';
$servers = array(
    'tcp://{redis_instance_address}:{port}'
);
$options = array('cluster' => 'redis');
$client = new Predis\Client($servers, $options);
$client->set('foo', 'bar');
$value = $client->get('foo');
echo $value;
?>
```

{redis_instance_address} indicates the actual IP address or domain name of the DCS instance and *{port}* is the actual port number of DCS instance. For details about how to obtain the IP address/domain name and port, see [Step 1](#). Change the IP address/domain name and port as required. *{password}* indicates the password used to log in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation. If password-free access is required, delete the line that contains "password".

Step 7 Run the **php redis.php** command to access the DCS instance.

----End

3.2.2.8 Node.js

Access a DCS Redis instance through Node.js on an ECS in the same VPC. For more information about how to use other Redis clients, visit [the Redis official website](#).

NOTE

The operations described in this section apply only to single-node, master/standby, and Proxy Cluster instances. To use Node.js to connect to a Redis Cluster instance, see [Node.js Redis client description](#).

Prerequisites

- A DCS Redis instance has been created and is in the **Running** state.
- An ECS has been created. For details about how to create an ECS, see *Elastic Cloud Server User Guide*.
- If the ECS runs the Linux OS, ensure that the GCC compilation environment has been installed on the ECS.

Procedure

- **For client servers running Ubuntu (Debian series):**

Step 1 View the IP address/domain name and port number of the DCS Redis instance to be accessed.

For details, see [Viewing Details of a DCS Instance](#).

Step 2 Log in to the ECS.

Step 3 Install Node.js.

```
apt install nodejs-legacy
```

If the preceding command does not work, run the following commands:

```
wget https://nodejs.org/dist/v0.12.4/node-v0.12.4.tar.gz --no-check-certificate
```

```
tar -xvf node-v4.28.5.tar.gz
```

```
cd node-v4.28.5
```

```
./configure
```

```
make
```

```
make install
```

 **NOTE**

After the installation is complete, run the **node --version** command to query the Node.js version to check whether the installation is successful.

Step 4 Install the node package manager (npm).

```
apt install npm
```

Step 5 Install the Redis client ioredis.

```
npm install ioredis
```

Step 6 Edit the sample script for connecting to a DCS instance.

Add the following content to the **ioredisdemo.js** script, including information about connection and data reading.

```
var Redis = require('ioredis');
var redis = new Redis({
  port: 6379, // Redis port
  host: '192.168.0.196', // Redis host
  family: 4, // 4 (IPv4) or 6 (IPv6)
  password: '*****',
  db: 0
});
redis.set('foo', 'bar');
redis.get('foo', function (err, result) {
  console.log(result);
});
// Or using a promise if the last argument isn't a function
redis.get('foo').then(function (result) {
  console.log(result);
});
// Arguments to commands are flattened, so the following are the same:
redis.sadd('set', 1, 3, 5, 7);
redis.sadd('set', [1, 3, 5, 7]);
// All arguments are passed directly to the redis server:
redis.set('key', 100, 'EX', 10);
```

host indicates the example IP address/domain name of DCS instance and *port* indicates the port number of DCS instance. For details about how to obtain the IP address/domain name and port, see [Step 1](#). Change the IP address/domain name

and port as required. ********* indicates the password used for logging in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation.

Step 7 Run the sample script to access the chosen DCS instance.

```
node ioredisdemo.js
```

```
----End
```

- **For client servers running CentOS (Red Hat series):**

Step 1 View the IP address/domain name and port number of the DCS Redis instance to be accessed.

For details, see .

Step 2 Log in to the ECS.

Step 3 Install Node.js.

```
yum install nodejs
```

If the preceding command does not work, run the following commands:

```
wget https://nodejs.org/dist/v0.12.4/node-v0.12.4.tar.gz --no-check-certificate
```

```
tar -xvf node-v0.12.4.tar.gz
```

```
cd node-v0.12.4
```

```
./configure
```

```
make
```

```
make install
```

NOTE

After the installation is complete, run the **node --version** command to query the Node.js version to check whether the installation is successful.

Step 4 Install npm.

```
yum install npm
```

Step 5 Install the Redis client ioredis.

```
npm install ioredis
```

Step 6 Edit the sample script for connecting to a DCS instance.

Add the following content to the **ioredisdemo.js** script, including information about connection and data reading.

```
var Redis = require('ioredis');  
var redis = new Redis({  
  port: 6379, // Redis port  
  host: '192.168.0.196', // Redis host  
  family: 4, // 4 (IPv4) or 6 (IPv6)  
  password: '*****',  
  db: 0  
});  
redis.set('foo', 'bar');
```

```
redis.get('foo', function (err, result) {
  console.log(result);
});
// Or using a promise if the last argument isn't a function
redis.get('foo').then(function (result) {
  console.log(result);
});
// Arguments to commands are flattened, so the following are the same:
redis.sadd('set', 1, 3, 5, 7);
redis.sadd('set', [1, 3, 5, 7]);
// All arguments are passed directly to the redis server:
redis.set('key', 100, 'EX', 10);
```

host indicates the example IP address/domain name of DCS instance and *port* indicates the port number of DCS instance. For details about how to obtain the IP address/domain name and port, see [Step 1](#). Change the IP address/domain name and port as required. ******* indicates the password used for logging in to the chosen DCS Redis instance. This password is defined during DCS Redis instance creation.

Step 7 Run the sample script to access the chosen DCS instance.

```
node ioredisdemo.js
```

```
----End
```

3.2.3 Accessing a DCS Redis 4.0 or 5.0 Instance on the Console

Access a DCS Redis instance through Web CLI. This function is supported only by DCS Redis 4.0 and 5.0 instances, and not by DCS Redis 3.0 instances.

NOTE

- Do not enter sensitive information in Web CLI to avoid disclosure.
- Keys and values cannot contain spaces.
- If the value is empty, **nil** is returned after the **GET** command is executed.

Prerequisites

The DCS Redis 4.0 or 5.0 instance you want to access through Web CLI is in the **Running** state.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 In the row containing the desired instance, choose **More > Connect to Redis** to go to the Web CLI login page.

Step 5 Enter the password of the DCS instance. On Web CLI, select the current Redis database, enter a Redis command in the command box, and press **Enter**.

```
----End
```


3.2.4 Accessing a DCS Memcached Instance

Access a DCS Memcached instance using telnet on an ECS in the same VPC.

Prerequisites

- The DCS Memcached instance you want to access is in the **Running** state.
- An ECS has been created on which the client has been installed. For details on how to create ECSs, see the *Elastic Cloud Server User Guide*.

NOTE

1. An ECS can communicate with a DCS instance that belongs to the same VPC and is configured with the same security group.
 2. If the ECS and DCS instance are not in the same VPC, you can establish a VPC peering connection to achieve network connectivity between the ECS and DCS instance. For details, see [Does DCS Support Cross-VPC Access?](#)
 3. If different security groups have been configured for the ECS and DCS instance, you can set security group rules to achieve network connectivity between the ECS and DCS instance. For details, see [Security Group Configurations](#).
- All annotations in example code have been deleted.
 - All command lines and code blocks are UTF-8 encoded. Using another encoding scheme will cause compilation problems or even command failures.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 On the **Cache Manager** page, click the name of the DCS Memcached instance you want to access. Obtain the IP address and port number of the instance.

Step 5 Access the chosen DCS Memcached instance.

1. Log in to the ECS.
2. Run the following command to check whether telnet is installed on the ECS:

which telnet

If the directory in which the telnet is installed is displayed, telnet has been installed on the ECS. If the client installation directory is not displayed, install the telnet manually.

NOTE

- If telnet has not been installed in Linux, run the **yum -y install telnet** command to install it.
 - In the Windows OS, choose **Start > Control Panel > Programs > Programs and Features > Turn Windows features on or off**, and enable telnet.
3. Run the following command to access the chosen DCS Memcached instance:
telnet {ip} {port}

In this command: *{ip}* indicates the IP address of the DCS Memcached instance. *{port}* indicates the port number of the DCS Memcached instance. Both the IP address and the port number are obtained in [Step 4](#).

When you have successfully accessed the chosen DCS Memcached instance, information similar to the following is displayed:

```
Trying XXX.XXX.XXX.XXX...  
Connected to XXX.XXX.XXX.XXX.  
Escape character is '^'.
```

 **NOTE**

- If **Password-protected** is not enabled for the instance, run the following commands directly after the instance is accessed successfully.
- If **Password-protected** is enabled for the instance, attempts to perform operations on the instance will result in the message "ERROR authentication required", indicating that you do not have the required permissions. In this case, enter **auth *username@password*** to authenticate first. *username* and *password* are that used for accessing the DCS Memcached instance.

Example commands for using the DCS Memcached instance (lines in bold are the commands and the other lines are the command output):

```
set hello 0 0 6  
world!  
STORED  
get hello  
VALUE hello 0 6  
world!  
END
```

----End

3.3 Viewing Details of a DCS Instance

On the DCS console, you can view DCS instance details.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Click an instance to go to the instance details page.




Step 5 Search for DCS instances using any of the following methods:


- Search by keyword.
Enter a keyword to search.
- Select attributes and enter their keywords to search.
Currently, you can search by name, ID, connection address (IP address:port number), AZ, status, instance type, cache engine, and CPU.

For more information on how to search, click the question mark to the right of the search box.

Step 6 On the DCS instance list, click the name of a DCS instance to display more details about it. [Table 3-1](#) describes the parameters.

Table 3-1 Parameters on the Basic Information page of a DCS instance

Section	Parameter	Description
Instance Details	Name	Name of the chosen instance. To modify the instance name, click the  icon.
	Status	State of the chosen instance.
	ID	ID of the chosen instance.
	Cache Engine	Cache engine used by the DCS instance, which can be Redis or Memcached. If the cache engine is Redis, it is followed by the version number, for example, Redis 3.0.
	Instance Type	Type of the selected instance. Currently, supported types include single-node, master/standby, Proxy Cluster, and Redis Cluster.
	Cache Size	Specification of the chosen instance.
	Used/ Available Memory (MB)	The used memory space and maximum available memory space of the chosen instance. The used memory space includes: <ul style="list-style-type: none"> • Size of data stored on the DCS instance • Size of Redis-server buffers (including client buffer and repl-backlog) and internal data structures
	CPU	CPU of the DCS instance. This parameter is displayed only for DCS Redis instances.
	Created	Time at which the chosen instance started to be created.
	Run	Time at which the instance was created.
	Maintenance	Time range for any scheduled maintenance activities on cache nodes of this DCS instance. To modify the time window, click the  icon.
	Description	Description of the chosen DCS instance. To modify the description, click the  icon.
Connection	Password Protected	Currently, password-protected access and password-free access are supported.
	IP Address	IP address and port number of the chosen instance.
Network	AZ	Availability zone in which the cache node running the selected DCS instance resides.
	VPC	VPC in which the chosen instance resides.
	Subnet	Subnet in which the chosen instance resides.

Section	Parameter	Description
	Security Group	Security group that controls access to the chosen instance. To modify the security group, click the  icon. This parameter is displayed only for DCS Redis 3.0 instances. DCS for Redis 4.0 and 5.0 are based on VPC endpoints and do not support security groups.
Instance Topology	-	Hover the mouse pointer over an instance to view its metrics, or click the icon of an instance to view its historical metrics. Topologies are supported only for master/standby and cluster instances.

----End

4 Operation Guide

4.1 Operating DCS Instances

4.1.1 Modifying DCS Instance Specifications

On the DCS console, you can scale a DCS Redis or Memcached instance to a larger or smaller capacity.

 **NOTE**

- **Modify instance specifications during off-peak hours.**
- You can only change the instance type of single-node or master/standby DCS Redis 3.0 instances.
- If your DCS instances are too old to support scaling, contact technical support to upgrade the instances.
- Services may be interrupted for seconds during the modification. Therefore, services connected to Redis must support reconnection.

Change of the Instance Type

- **Supported instance type changes:**
 - From single-node to master/standby: Supported by Redis 3.0 and Memcached, and not by Redis 4.0 and 5.0.
 - From master/standby to Proxy Cluster: Supported by Redis 3.0, and not by Redis 4.0 and 5.0.
If the data of a master/standby DCS Redis 3.0 instance is stored in multiple databases, or in non-DB0 databases, the instance cannot be changed to the Proxy Cluster type. A master/standby instance can be changed to the Proxy Cluster type only if its data is stored only on DB0.
 - From cluster types to other types: Not supported.
- **Impact of instance type changes:**
 - From single-node to master/standby for a DCS Redis 3.0 instance:
The instance cannot be connected for several seconds and remains read-only for about 1 minute.

- From master/standby to Proxy Cluster for a DCS Redis 3.0 instance:
The instance cannot be connected and remains read-only for 5 to 30 minutes.

Scaling

- **The following table lists scaling options supported by different DCS instances.**

Table 4-1 Scaling options supported by different DCS instances

Cache Engine	Single-Node	Master/Standby	Redis Cluster	Proxy Cluster
Redis 3.0	Scaling up/down	Scaling up/down	N/A	Scaling up
Redis 4.0	Scaling up/down	Scaling up/down	Scaling up/down	N/A
Redis 5.0	Scaling up/down	Scaling up/down	Scaling up/down	N/A
Memcached	Scaling up/down	Scaling up/down	N/A	N/A

NOTE

If the reserved memory of a DCS Redis 3.0 or Memcached instance is insufficient, the scaling may fail when the memory is used up.


- **Impact of scaling:**
 - Single-node and master/standby
 - A DCS Redis 4.0 or 5.0 instance will be disconnected for several seconds and remain read-only for about 1 minute.
 - A DCS Redis 3.0 instance will be disconnected and remain read-only for 5 to 30 minutes.
 - For scaling up, only the memory of the instance is expanded. The CPU processing capability is not improved.
 - Data of single-node instances may be lost because they do not support data persistence. After scaling, check whether the data is complete and import data if required. If there is important data, use a migration tool to migrate the data to other instances for backup.
 - Backup records of master/standby instances cannot be restored after scaling down.
 - Cluster
 - If the shard quantity is not decreased, the instance can always be connected, but the CPU usage will increase, compromising performance by up to 20%, and the latency will increase during data

migration. During scaling up, new Redis Server nodes are added, and data is automatically balanced to the new nodes.

- Nodes will be deleted if the shard quantity decreases. To prevent disconnection, ensure that the deleted nodes are not directly referenced in your application.
- Ensure that the used memory of each node is less than 70% of the maximum memory per node of the new flavor. Otherwise, you cannot perform the scale-in.
- If the memory becomes full during scaling due to a large amount of data being written, scaling will fail.
- Scaling involves data migration. The latency for accessing the key being migrated increases. For a Redis Cluster instance, ensure that the client can properly process the **MOVED** and **ASK** commands. Otherwise, requests will fail.
- Before scaling, perform cache analysis to ensure that no big keys (\geq 512 MB) exist in the instance. Otherwise, scaling may fail.
- Backup records created before scaling cannot be restored.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Choose **More > Modify Specifications** in the row containing the DCS instance.

Step 5 On the **Modify Specifications** page, select the desired specification and click Next.

Step 6 Click **Submit**.

On the displayed **Background Tasks** page, view the modification status. For more information, see [Viewing Background Tasks](#).

Specification modification of a single-node or master/standby DCS instance takes about 5 to 30 minutes to complete, while that of a cluster DCS instance takes a longer time. After an instance is successfully modified, it changes to the **Running** state.

 **NOTE**

- If the specification modification of a single-node DCS instance fails, the instance is temporarily unavailable for use. The specification remains unchanged. Some management operations (such as parameter configuration and specification modification) are temporarily not supported. After the specification modification is completed in the backend, the instance changes to the new specification and becomes available for use again.
- If the specification modification of a master/standby or cluster DCS instance fails, the instance is still available for use with its original specifications. Some management operations (such as parameter configuration, backup, restoration, and specification modification) are temporarily not supported. Remember not to read or write more data than allowed by the original specifications; otherwise, data loss may occur.
- After the specification modification is successful, the new specification of the instance takes effect.

----End

4.1.2 Restarting DCS Instances

On the DCS console, you can restart one or multiple DCS instances at a time.

NOTICE

- After a single-node DCS instance is restarted, data will be deleted from the instance.
 - While a DCS instance is restarting, it cannot be read from or written to.
 - An attempt to restart a DCS instance while it is being backed up may result in a failure.
-

Prerequisites

The DCS instances you want to restart are in the **Running** or **Faulty** state.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 On the **Cache Manager** page, select one or more DCS instances you want to restart.

Step 5 Click **Restart** above the DCS instance list.

Step 6 In the displayed dialog box, click **Yes**.

It takes 1 to 30 minutes to restart DCS instances. After DCS instances are restarted, their status changes to **Running**.

 **NOTE**

- By default, only the instance process is restarted. If you select **Force restart** for a DCS Redis 3.0 or Memcached instance, its VM will be restarted. **Force restart** is not supported by DCS Redis 4.0 or later instances.
- To restart a single instance, you can also click **Restart** in the same row as the instance.
- The time required for restarting a DCS instance depends on the cache size of the instance.

----End

4.1.3 Deleting DCS Instances

On the DCS console, you can delete one or multiple DCS instances at a time. You can also delete all instance creation tasks that have failed to run.

NOTICE

- After a DCS instance is deleted, the instance data will also be deleted without backup. In addition, any backup data of the instance will be deleted. Therefore, download the backup files of the instance for permanent storage before deleting the instance.
 - If the instance is in cluster mode, all cluster nodes will be deleted.
-

Prerequisites

- The DCS instances you want to delete have been created.
- The DCS instances you want to delete are in the **Running** or **Faulty** state.

Procedure

Deleting DCS Instances

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 On the **Cache Manager** page, select one or more DCS instances you want to delete.

DCS instances in the **Creating, Restarting, Upgrading, Resizing, Clearing data, Backing up, or Restoring** state cannot be deleted.

Step 5 Choose **More > Delete** above the instance list.

Step 6 In the displayed dialog box, click **Yes**.

It takes 1 to 30 minutes to delete DCS instances.

 **NOTE**

To delete a single instance, choose **Operation > More > Delete** in the same row as the instance.

----End

Deleting Instance Creation Tasks That Have Failed to Run

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 If there are DCS instances that have failed to be created, **Instance Creation Failures** is displayed above the instance list.

Step 5 Click the icon or the number of failed tasks next to **Instance Creation Failures**.
The **Instance Creation Failures** dialog box is displayed.

Step 6 Choose failed instance creation tasks to delete.

- To delete all failed tasks, click **Delete All** above the task list.
- To delete a single failed task, click **Delete** in the same row as the task.

----End

4.1.4 Performing a Master/Standby Switchover for a DCS Instance

On the DCS console, you can manually switch the master and standby nodes of a DCS instance. This operation is used for special purposes, for example, releasing all service connections or terminating ongoing service operations.

Only master/standby instances support a master/standby node switchover.


NOTICE

- Services may be interrupted for up to 10 seconds during the switchover. Before performing a switchover, ensure that your application supports reconnection.
 - During a master/standby node switchover, a large amount of resources will be consumed for data synchronization between the master and standby nodes. You are advised to perform this operation during off-peak hours.
 - Data of the maser and standby nodes is synchronized asynchronously. Therefore, a small amount of data that is being operated on during the switchover may be lost.
-

Prerequisites

The DCS instance for which you want to perform a master/standby node switchover is in the **Running** state.

Procedure


- Step 1** Log in to the DCS console.
 - Step 2** Click  in the upper left corner and select a region and a project.
 - Step 3** In the navigation pane, choose **Cache Manager**.
 - Step 4** In the **Operation** column of the instance, choose **More > Master/Standby Switchover**.
- End

4.1.5 Clearing DCS Instance Data

On the DCS console, you can clear data only for DCS Redis 4.0 and 5.0 instances.

Clearing instance data cannot be undone and cleared data cannot be recovered. Exercise caution when performing this operation.



Procedure

- Step 1** Log in to the DCS console.
 - Step 2** Click  in the upper left corner and select a region and a project.
 - Step 3** In the navigation pane, choose **Cache Manager**.
 - Step 4** Select one or more DCS2.0 instances to clear.
 - Step 5** Click **Clear data** above the instance list.
 - Step 6** In the displayed dialog box, click **Yes**.
- End

4.1.6 Exporting DCS Instance List

On the DCS console, you can export DCS instance information in full to an Excel file.

Procedure

- Step 1** Log in to the DCS console.
- Step 2** Click  in the upper left corner and select a region and a project.
- Step 3** In the navigation pane, choose **Cache Manager**.
- Step 4** Click the search bar. In the displayed drop-down list, select the filter criteria to query the desired DCS instances.
- Step 5** Click  above the instance list.

Click the export result displayed in the lower left corner of the page.


----End

4.1.7 Command Renaming

After creating a DCS Redis 4.0 or 5.0 instance, you can rename the following critical commands: **COMMAND**, **KEYS**, **FLUSHDB**, **FLUSHALL**, and **HGETALL**.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 In the **Operation** column of an instance, choose **More > Command Renaming**.

Step 5 Select a command, enter a new name, and click **OK**.

NOTE

- You can rename multiple commands at a time.
- The new command names will take effect only after you restart the instance. Remember the new command names because they will not be displayed on the console for security purposes.
- To use the original name of a command, rename the command again.
- The new name must contain at least four characters.

----End

4.2 Managing DCS Instances

4.2.1 Configuration Notice

- In most cases, different DCS instance management operations cannot proceed concurrently. If you initiate a new management operation while the current operation is in progress, the DCS console prompts you to initiate the new operation again after the current operation is complete. DCS instance management operations include:
 - Creating a DCS instance
 - Configuring parameters
 - Restarting a DCS instance
 - Changing the instance password
 - Resetting the instance password
 - Scaling, backing up, or restoring an instance
- You can restart a DCS instance while it is being backed up, but the backup task will be forcibly interrupted and is likely to result in a backup failure.

NOTICE

In the event that a cache node of a DCS instance is faulty:

- The instance remains in the **Running** state and you can continue to read from and write to the instance. This is achieved thanks to the high availability of DCS.
- Cache nodes can recover from internal faults automatically. Manual fault recovery is also supported.
- Certain operations (such as parameter configuration, password change or resetting, backup, restoration, and specification modification) in the management zone are not supported during fault recovery. You can contact technical support or perform these operations after the cache nodes recover from faults.


4.2.2 Modifying Configuration Parameters

You can modify the configuration parameters of your DCS instance to optimize DCS performance based on your requirements.

For example, if you do not need data persistence, set **appendonly** to **no**.

After the instance configuration parameters are modified, the modification takes effect immediately without the need to manually restart the instance. For a cluster instance, the modification takes effect on all shards.

Procedure

- Step 1** Log in to the DCS console.
- Step 2** Click  in the upper left corner and select a region and a project.
- Step 3** In the navigation pane, choose **Cache Manager**.
- Step 4** On the **Cache Manager** page, click the name of the DCS instance you want to configure.
- Step 5** Choose **Instance Configuration > Parameters**.
- Step 6** On the **Parameters** tab page, click **Modify**.
- Step 7** Modify parameters based on your requirements.

[Table 4-2](#) and [Table 4-3](#) describe the parameters. In most cases, retain the default values.

Table 4-2 DCS Redis instance configuration parameters

Parameter	Description	Value Range	Default Value
timeout	The maximum amount of time (in seconds) a connection between a client and the DCS instance can be allowed to remain idle before the connection is terminated. A setting of 0 means that this function is disabled.	0–7200 seconds	0
appendfsync	Controls how often fsync() transfers cached data to the disk. Note that some OSs will perform a complete data transfer but some others only make a "best-effort" attempt. There are three settings: no: fsync() is never called. The OS will flush data when it is ready. This mode offers the highest performance. always: fsync() is called after every write to the AOF. This mode is very slow, but also very safe. everysec: fsync() is called once per second. This mode provides a compromise between safety and performance.	<ul style="list-style-type: none"> • no • always • everysec 	everysec
appendonly	Indicates whether to log each modification of the instance. By default, data is written to disks asynchronously in Redis. If this function is disabled, recently-generated data might be lost in the event of a power failure. Options: yes: enabled no: disabled	<ul style="list-style-type: none"> • yes • no 	yes

Parameter	Description	Value Range	Default Value
client-output-buffer-limit-slave-soft-seconds	Number of seconds that the output buffer remains above client-output-buffer-slave-soft-limit before the client is disconnected.	0-60	60
client-output-buffer-slave-hard-limit	Hard limit (in bytes) on the output buffer of replica clients. Once the output buffer exceeds the hard limit, the client is immediately disconnected.	Depends on the instance type and specifications.	Depends on the instance type and specifications.
client-output-buffer-slave-soft-limit	Soft limit (in bytes) on the output buffer of replica clients. Once the output buffer exceeds the soft limit and continuously remains above the limit for the time specified by the client-output-buffer-limit-slave-soft-seconds parameter, the client is disconnected.	Depends on the instance type and specifications.	Depends on the instance type and specifications.

Parameter	Description	Value Range	Default Value
maxmemory-policy	<p>The deletion policy to apply when the maxmemory limit is reached. Options:</p> <p>volatile-lru: Evict keys by trying to remove the less recently used (LRU) keys first, but only among keys that have an expire set. (Recommended)</p> <p>allkeys-lru: Evict keys by trying to remove the LRU keys first.</p> <p>volatile-random: evict keys randomly, but only evict keys with an expire set.</p> <p>allkeys-random: Evict keys randomly.</p> <p>volatile-ttl: Evict keys with an expire set, and try to evict keys with a shorter time to live (TTL) first.</p> <p>noeviction: Do not delete any keys and only return errors when the memory limit was reached.</p> <p>volatile-lfu: Evict keys by trying to remove the less frequently used (LFU) keys first, but only among keys that have an expire set.</p> <p>allkeys-lfu: Evict keys by trying to remove the LFU keys first.</p>	Depends on the instance version.	Depends on the instance version and type.
lua-time-limit	Maximum time allowed for executing a Lua script (in milliseconds).	100-5000	5,000
master-read-only	Sets the instance to be read-only. All write operations will fail.	<ul style="list-style-type: none"> • yes • no 	no

Parameter	Description	Value Range	Default Value
maxclients	The maximum number of clients allowed to be concurrently connected to a DCS instance.	Depends on the instance type and specifications.	Depends on the instance type and specifications.
proto-max-bulk-len	Maximum size of a single element request (in bytes).	1,048,576–536,870,912	536,870,912
repl-backlog-size	The replication backlog size (bytes). The backlog is a buffer that accumulates replica data when replicas are disconnected from the master. When a replica reconnects, a partial synchronization is performed to synchronize the data that was missed while replicas were disconnected.	16,384–1,073,741,824	1,048,576
repl-backlog-ttl	The amount of time, in seconds, before the backlog buffer is released, starting from the last a replica was disconnected. The value 0 indicates that the backlog is never released.	0–604,800	3,600
repl-timeout	Replication timeout (in seconds).	30–3,600	60
hash-max-ziplist-entries	Hashes are encoded using a memory efficient data structure when the number of entries in hashes is less than the value of this parameter.	1–10,000	512

Parameter	Description	Value Range	Default Value
hash-max-ziplist-value	Hashes are encoded using a memory efficient data structure when the biggest entry in hashes does not exceed the length threshold indicated by this parameter.	1-10,000	64
set-max-intset-entries	When a set is composed of just strings that happen to be integers in radix 10 in the range of 64 bit signed integers, sets are encoded using a memory efficient data structure.	1-10,000	512
zset-max-ziplist-entries	Sorted sets are encoded using a memory efficient data structure when the number of entries in sorted sets is less than the value of this parameter.	1-10,000	128
zset-max-ziplist-value	Sorted sets are encoded using a memory efficient data structure when the biggest entry in sorted sets does not exceed the length threshold indicated by this parameter.	1-10,000	64

Parameter	Description	Value Range	Default Value
latency-monitor-threshold	<p>Threshold time in latency monitoring. Unit: millisecond.</p> <p>Set to 0: Latency monitoring is disabled.</p> <p>Set to more than 0: All with at least this many milliseconds of latency will be logged.</p> <p>By running the LATENCY command, you can perform operations related to latency monitoring, such as obtaining statistical data, and configuring and enabling latency monitoring.</p>	0-86,400,000 ms	0

Parameter	Description	Value Range	Default Value
notify-keyspace-events	<p>Controls which keyspace events notifications are enabled for. If the value is an empty string, this function is disabled. A combination of different values can be used to enable notifications for multiple event types. Possible values:</p> <p>K: Keyspace events, published with the __keyspace@__ prefix.</p> <p>E: Keyevent events, published with __keyevent@__ prefix</p> <p>g: Generic commands (non-type specific) such as DEL, EXPIRE, and RENAME</p> <p>\$: String commands</p> <p>l: List commands</p> <p>s: Set commands</p> <p>h: Hash commands</p> <p>z: Sorted set commands</p> <p>x: Expired events (events generated every time a key expires)</p> <p>e: Evicted events (events generated when a key is evicted for maxmemory)</p> <p>A: an alias for "g\$lshzxe"</p> <p>The parameter value must contain either K or E. A cannot be used together with any of the characters in "g\$lshzxe". For example, the value Kl means that Redis will notify Pub/Sub clients about keyspace events and list commands. The value AKE means Redis will notify Pub/Sub clients about all events.</p>	See the parameter description.	Ex

Parameter	Description	Value Range	Default Value
slowlog-log-slower-than	Redis records queries that exceed a specified execution time. slowlog-log-slower-than is the maximum time allowed, in microseconds, for command execution. If this threshold is exceeded, Redis will record the query.	0–1,000,000	10,000
slowlog-max-len	The maximum allowed number of slow queries that can be logged. Slow query log consumes memory, but you can reclaim this memory by running the SLOWLOG RESET command.	0–1000	128

 NOTE

1. For more information about the parameters described in [Table 4-2](#), visit <https://redis.io/topics/memory-optimization>.
2. The **latency-monitor-threshold** parameter is usually used for fault location. After locating faults based on the latency information collected, change the value of **latency-monitor-threshold** to **0** to avoid unnecessary latency.
3. More about the **notify-keyspace-events** parameter:
 - The parameter setting must contain at least a **K** or **E**.
 - **A** is an alias for "g\$lshzxe" and cannot be used together with any of the characters in "g\$lshzxe".
 - For example, the value **KI** means that Redis will notify Pub/Sub clients about keyspace events and list commands. The value **AKE** means Redis will notify Pub/Sub clients about all events.

Table 4-3 DCS Memcached instance configuration parameters

Parameter	Description	Value Range	Default Value
timeout	The maximum amount of time (in seconds) a connection between a client and the DCS instance can be allowed to remain idle before the connection is terminated. A setting of 0 means that this function is disabled.	0-7200 seconds	0
maxclients	The maximum number of clients allowed to be concurrently connected to a DCS instance.	1000-10,000	10,000
maxmemory-policy	The policy applied when the maxmemory limit is reached. For more information about this parameter, see https://redis.io/topics/lru-cache .	volatile-lru allkeys-lru volatile-random allkeys-random volatile-ttl noeviction	noeviction
reserved-memory-percent	Percentage of the maximum available memory reserved for background processes, such as data persistence and replication.	0-80	30

Step 8 After you have finished setting the parameters, click **Save**.

Step 9 Click **Yes** to confirm the modification.

----End

Typical Scenarios of Configuring Parameters

The following describes how to change the value of the **appendonly** parameter:

- If Redis is used as the cache and services are insensitive to Redis data losses, disable instance persistence to improve performance. In this case, change the value of **appendonly** to **no**. For details, see [Procedure](#).
- If Redis is used as the database or services are sensitive to Redis data losses, enable instance persistence. In this case, change the value of **appendonly** to **yes**. For details, see [Procedure](#). After instance persistence is enabled, you need to consider the frequency of writing Redis cache data to disks and the impact on the Redis performance. You can use this parameter together with the **appendfsync** parameter. There are three modes of calling fsync():

- **no**: fsync() is never called. The OS will flush data when it is ready. This mode offers the highest performance.
- **always**: fsync() is called after every write to the AOF. This mode is very slow, but also very safe.
- **everysec**: fsync() is called once per second, ensuring both data security and performance.

 **NOTE**

Currently, the **appendonly** and **appendfsync** parameters can be modified on the console only for master/standby and Redis Cluster instances.

4.2.3 Modifying Maintenance Time Window

On the DCS console, after creating a DCS instance, you can modify the maintenance time window of the DCS instance on the instance's **Basic Information** page.

Prerequisites

At least one DCS instance has been created.


Procedure



Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Click the name of the DCS instance for which you want to modify the maintenance time window.

Step 5 Click the **Basic Information** tab. In the **Instance Details** area, click the  icon next to the **Maintenance** parameter.

Step 6 Select a new maintenance time window from the drop-down list. Click  to save the modification or  to discard the modification.

The modification will take effect immediately, that is, the new maintenance time window will appear on the **Basic Information** tab page immediately.

----End

4.2.4 Modifying the Security Group





On the DCS console, after creating a DCS instance, you can modify the security group of the DCS instance on the instance's **Basic Information** page.

You can modify the security groups of DCS Redis 3.0 instances but cannot modify those of DCS Redis 4.0 or 5.0 instances.

Prerequisites

At least one DCS instance has been created.

Procedure

- Step 1** Log in to the DCS console.
- Step 2** Click  in the upper left corner and select a region and a project.
- Step 3** In the navigation pane, choose **Cache Manager**.
- Step 4** Click the name of the DCS instance for which you want to modify the security group.
- Step 5** Click the **Basic Information** tab. In the **Network** area, click  next to the **Security Group** parameter.
- Step 6** Select a new security group from the drop-down list. Click  to save the modification or  to discard the modification.

NOTE

Only the security groups that have been created can be selected from the drop-down list. If you need to create a security group, follow the procedure described [Security Group Configurations](#).



The modification will take effect immediately, that is, the new maintenance time window will appear on the **Basic Information** tab page immediately.


----End

4.2.5 Viewing Background Tasks

After you initiate certain instance operations such as modifying instance specifications and changing or resetting a password, a background task will start for the operation. On the DCS console, you can view the background task status and clear task information by deleting task records.

Procedure

- Step 1** Log in to the DCS console.
- Step 2** Click  in the upper left corner and select a region and a project.
- Step 3** In the navigation pane, choose **Cache Manager**.
- Step 4** Click the name of the DCS instance whose background task you want to manage.
- Step 5** Click the **Background Tasks** tab.
A list of background tasks is displayed.
- Step 6** Click , specify **Start Date** and **End Date**, and click **OK** to view tasks started in the corresponding time segment.

- Click  to refresh the task status.
- To clear the record of a background task, click **Delete** in the **Operation** column.

 **NOTE**

You can only delete the records of tasks in the **Successful** or **Failed** state.

----End

4.2.6 Viewing Data Storage Statistics of a DCS Redis 3.0 Proxy Cluster Instance

You can view the data storage statistics of all nodes of a DCS Redis 3.0 Proxy Cluster instance. If data storage is unevenly distributed across nodes, you can scale up the instance or clear data.


You can only view data storage statistics of DCS Redis 3.0 Proxy Cluster instances. Instances of other types, for example, master/standby, only have one node, and you can view the used memory on the instance details page.

 **NOTE**

A Redis Cluster instance has multiple storage nodes. You can check the data storage statistics of a Redis Cluster instance in its Redis Server monitoring data.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Click the name of a DCS Redis cluster instance to view the basic information.

Step 5 Click the **Node Management** tab.

The data volume of each node in the cluster instance is displayed.

When the data storage capacity of a node in a cluster is used up, you can scale up the instance according to [Modifying DCS Instance Specifications](#).

----End

4.2.7 Managing Tags

Tags facilitate DCS instance identification and management.

You can add tags to an instance when creating it or add, modify, or delete tags on the details page of a created instance. Each instance can have a maximum of 10 tags.


A tag consists of a tag key and a tag value. [Table 4-4](#) lists the tag key and value requirements.

Table 4-4 Tag key and value requirements

Parameter	Requirement
Tag key	<ul style="list-style-type: none"> • Cannot be left blank. • Must be unique for the same instance. • Can contain a maximum of 36 characters. • Cannot contain the following characters: =*<>\, / • Cannot start or end with a space.
Tag value	<ul style="list-style-type: none"> • Can contain a maximum of 43 characters. • Cannot contain the following characters: =*<>\, / • Cannot start or end with a space.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Click the name of an instance.

Step 5 Click the **Tags** tab.

View the tags of the instance.

Step 6 Perform the following operations as required:

- Add a tag
 - a. Click **Add Tag**.
If you have created predefined tags, select a predefined pair of tag key and value. To view predefined tags or create tags, click **View predefined tags**. You will be directed to the TMS console.
You can also create new tags by entering **Tag key** and **Tag value**.
 - b. Click **OK**.
- Modify a tag
In the row containing the tag to be modified, click **Edit** in the **Operation** column. Enter the new tag value and click **OK**.
- Delete a tag
In the row containing the tag to be deleted, click **Delete** in the **Operation** column. Then click **Yes**.

----End

4.2.8 Managing Shards and Replicas

This section describes how to query the shards and replicas of a DCS Redis 4.0 or 5.0 instance and how to manually promote a replica to master.

Currently, this function is supported only by cluster and master/standby DCS Redis 4.0 or 5.0 instances. DCS Redis 3.0 instances and single-node DCS Redis 4.0/5.0 instances do not support this function.

- A master/standby instance has only one shard with one master and one replica by default. You can view the sharding information on the **Shards and Replicas** page. To manually switch the master and replica roles, see [Performing a Master/Standby Switchover for a DCS Instance](#).
- A cluster instance has multiple shards. Each shard has one master and one replica by default. On the **Shards and Replicas** page, you can view the sharding information and manually switch the master and replica roles. For details about the number of shards corresponding to different instance specifications, see [Redis Cluster](#).

Promoting a Replica to Master

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**. The **Cache Manager** page is displayed.

Step 4 Click an instance.

Step 5 Click the **Shards and Replicas** tab.

The page displays all shards in the instance and the list of replicas of each shard.

Step 6 Click  to show all replicas of a shard.

Step 7 Click **Promote to Master** in the row containing another replica which is in the "Replica" role.

Step 8 Click **Yes**.

----End

4.2.9 Cache Analysis

By performing big key analysis and hot key analysis, you will have a picture of keys that occupy a large space and keys that are the most frequently accessed.

Notes on big key analysis:

- All DCS Redis instances support big key analysis.
- During big key analysis, all keys will be traversed. The larger the number of keys, the longer the analysis takes.
- Perform big key analysis during off-peak hours and avoid automatic backup periods.

- For a master/standby or cluster instance, the big key analysis is performed on the standby node, so the impact on the instance is minor. For a single-node instance, the big key analysis is performed on the only node of the instance and will reduce the instance access performance by up to 10%. Therefore, perform big key analysis on single-node instances during off-peak hours.
- A maximum of 100 big key analysis records (20 for Strings and 80 for Lists/Sets/Zsets/Hashes) are retained for each instance. When this limit is reached, the oldest records will be deleted to make room for new records. You can also manually delete records you no longer need.

Notes on hot key analysis:

- Only DCS Redis 4.0 and 5.0 instances support hot key analysis, and the **maxmemory-policy** parameter of the instances must be set to **allkeys-lfu** or **volatile-lfu**.
- During hot key analysis, all keys will be traversed. The larger the number of keys, the longer the analysis takes.
- Perform hot key analysis shortly after peak hours to ensure the accuracy of the analysis results.
- The hot key analysis is performed on the master node of each instance and will reduce the instance access performance by up to 10%.
- A maximum of 100 hot key analysis records are retained for each instance. When this limit is reached, the oldest records will be deleted to make room for new records. You can also manually delete records you no longer need.

NOTE

Perform big key and hot key analysis during off-peak hours to avoid 100% CPU usage.

Big Key Analysis Procedure

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Click the name of a DCS Redis instance.

Step 5 Click the **Cache Analysis** tab.

Step 6 On the **Big Key Analysis** tab page, manually perform big key analysis or schedule daily automatic analysis.

Step 7 After an analysis task completes, click **View** to view the analysis results.

You can view the analysis results of different data types.


NOTE

A maximum of 20 big key analysis records are retained for Strings and 80 are retained for Lists, Sets, Zsets, and Hashes.

----End

Hot Key Analysis Procedure

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Click the name of a DCS Redis instance.

Step 5 Click the **Cache Analysis** tab.

Step 6 On the **Hot Key Analysis** tab page, manually perform hot key analysis or schedule daily automatic analysis.

 **NOTE**

The default value of the **maxmemory-policy** parameter of a Redis 4.0 or 5.0 instance is **noeviction**. To perform hot key analysis, set this parameter to **allkeys-lfu** or **volatile-lfu**. If this parameter has already been set to **allkeys-lfu** or **volatile-lfu**, perform hot key analysis right away.

Step 7 After an analysis task completes, click **View** to view the analysis results.

The hot key analysis results are displayed.

 **NOTE**

The console displays a maximum of 100 hot key analysis records for each instance.

Table 4-5 Results of hot key analysis

Parameter	Description
Key	Name of a hot key.
Type	Type of a hot key, which can be string, hash, list, set, or sorted set.
Size	Size of the hot key value.
FREQ	Reflects the access frequency of a key within a specific period of time. FREQ is the logarithmic access frequency counter. The maximum value of FREQ is 255, which indicates 1 million access requests. After FREQ reaches 255 , it will no longer increment even if access requests continue to increase. FREQ will decrement by 1 for every minute during which the key is not accessed.
DataBase	Database where a hot key is located.

----End

4.2.10 Managing IP Address Whitelist


DCS helps you control access to your DCS instances in the following ways, depending on the deployment mode:

- To control access to Redis 3.0 and Memcached instances, you can use security groups. Whitelists are not supported. For details about how to configure a security group, see [Security Group Configurations](#).
- To control access to Redis 4.0 and 5.0 instances, you can use whitelists. Security groups are not supported.

The following describes how to manage whitelists of a Redis 4.0 or 5.0 instance to allow access only from whitelisted IP addresses. If no whitelists are added for the instance or the whitelist function is disabled, all IP addresses that can communicate with the VPC can access the instance.

Creating a Whitelist Group

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner of the management console and select a region.

 **NOTE**

Select the same region as your application service.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Click the name of a DCS instance.

Step 5 Click the **Whitelist** tab and then click **Create Whitelist Group**.

Step 6 In the **Create Whitelist Group** dialogue box, specify **Group Name** and **IP Address/Range**.

Table 4-6 Whitelist parameters

Parameter	Description	Example
Group Name	Whitelist group name of the instance. A maximum of four whitelist groups can be created for each instance.	DCS-test

Parameter	Description	Example
IP Address/ Range	A maximum of 20 IP addresses or IP address ranges can be added to an instance. Separate multiple IP addresses or IP address ranges with commas. Unsupported IP address and IP address range: 0.0.0.0 and 0.0.0/0.	10.10.10.1,10.10.10.10

Step 7 Click **OK**.

A whitelist group is automatically enabled for the instance once created. Only whitelisted IP addresses can access the instance.

 **NOTE**

- In the whitelist group list, click **Modify** to modify the IP addresses or IP address ranges in a group, and click **Delete** to delete a whitelist group.
- After whitelist has been enabled, you can click **Disable Whitelist** above the whitelist group list to allow all IP addresses connected to the VPC to access the instance.

----End

4.2.11 Viewing Redis Slow Queries

Redis logs queries that exceed a specified execution time. You can view the slow query log on the DCS console to identify performance issues.

For details about the commands, visit the [Redis official website](#).

Configure slow queries with the following parameters:


- **slowlog-log-slower-than**: The maximum time allowed, in microseconds, for command execution. If this threshold is exceeded, Redis will log the command. The default value is **10,000**. That is, if command execution exceeds 10 ms, the command will be logged.
- **slowlog-max-len**: The maximum allowed number of slow queries that can be logged. The default value is **128**. That is, if the number of slow queries exceeds 128, the earliest record will be deleted to make room for new ones.

For details about the configuration parameters, see [Modifying Configuration Parameters](#).

 **NOTE**

You can view the slow queries of a Proxy Cluster DCS Redis 3.0 instance only if the instance is created after October 14, 2019. If the instance was created earlier, submit a service ticket to upgrade it. The upgrade adds the slow query function to the console, and does not affect services.

Viewing Slow Queries on the Console

- Step 1** Log in to the DCS console.
- Step 2** Click  in the upper left corner and select a region and a project.
- Step 3** In the navigation pane, choose **Cache Manager**.
- Step 4** Click the name of a DCS instance.
- Step 5** Click **Slow Queries**.
- Step 6** Select a start date and an end date to view slow queries within the specified period.

 **NOTE**


For details about the commands, visit the [Redis official website](#).

----End

4.2.12 Viewing Redis Run Logs

You can create run log files on the DCS console to collect run logs of DCS Redis instances within a specified period. After the logs are collected, you can download the log files to view the logs.

Procedure

- Step 1** Log in to the DCS console.
- Step 2** Click  in the upper left corner and select a region and a project.
- Step 3** In the navigation pane, choose **Cache Manager**.
- Step 4** Click the name of a DCS instance.
- Step 5** Click the **Run Logs** tab.
- Step 6** Click **Create Log File**.

If the instance is the master/standby or cluster type, you can specify the shard and replica whose run logs you want to collect. If the instance is the single-node type, logs of the only node of the instance will be collected.

----End

4.2.13 Diagnosing an Instance

Scenario


If a fault or performance issue occurs, you can ask DCS to diagnose your instance to learn about the cause and impact of the issue and how to handle it.

Restrictions

- DCS Redis 3.0 and Memcached instances do not support diagnosis.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner of the management console and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Click the name of a DCS Redis instance.

Step 5 Click the **Instance Diagnosis** tab.

Step 6 Specify the tested object and time range, and click **Start Diagnosis**.

- **Tested Object:** You can select a single node or all nodes. By default, all nodes are tested.
- **Range:** You can specify up to 10 minutes before a point in time in the last 7 days.

Step 7 After the diagnosis is complete, you can view the result in the **Test History** list. If the result is abnormal, click **View Report** for details.

In the report, you can view the cause and impact of abnormal items and suggestions for handling them.

----End

4.3 Backing Up and Restoring DCS Instances

4.3.1 Overview

On the DCS console, you can back up and restore DCS instances.

Importance of DCS Instance Backup

There is a small chance that inconsistent data could exist in a DCS instance owing to service system exceptions or problems in loading data from persistence files. In addition, some systems demand not only high reliability but also data security, data restoration, and even permanent data storage.

Currently, data in DCS instances can be backed up to OBS. If a DCS instance becomes faulty, data in the instance can be restored from backup so that service continuity is not affected.

Backup Modes

DCS instances support the following backup modes:

- **Automated backup**
You can create a scheduled backup policy on the DCS console. Then, data in the chosen DCS instances will be automatically backed up at the scheduled time.

You can choose the days of the week on which scheduled backup will run. Backup data will be retained for a maximum of seven days. Backup data older than seven days will be automatically deleted.

The primary purpose of scheduled backups is to create complete data replicas of DCS instances so that the instance can be quickly restored if necessary.

- Manual backup

Backup requests can also be issued manually. Then, data in the chosen DCS instances will be permanently backed up to OBS. Backup data can be deleted manually.

Before performing high-risk operations, such as system maintenance or upgrade, back up DCS instance data.

Additional Information About Data Backup

- Instance type

- Redis: Only master/standby, Proxy Cluster, and Redis Cluster instances can be backed up and restored, while single-node instances cannot. However, you can export data of a single-node instance to an RDB file using `redis-cli`. For details, see [Can I Export Backup Data of DCS Redis Instances to RDB Files Using the Console?](#)
- Memcached: Only master/standby instances can be backed up and restored, while single-node instances cannot.

- Backup mechanisms

Instance data is persisted using the Redis Append Only Files (AOF) feature.

Backup tasks run on standby cache nodes. DCS instance data is backed up by compressing and storing the data persistence files from the standby cache node to OBS.

DCS checks instance backup policies once an hour. If a backup policy is matched, DCS runs a backup task for the corresponding DCS instance.

- Impact on DCS instances during backup

Backup tasks run on standby cache nodes, without incurring any downtime.

In the event of full-data synchronization or heavy instance load, it takes a few minutes to complete data synchronization. If instance backup starts before data synchronization is complete, the backup data will be slightly behind the data in the master cache node.

During instance backup, the standby cache node stops persisting the latest changes to disk files. If new data is written to the master cache node during backup, the backup file will not contain the new data.

- Backup time

It is advisable to back up instance data during off-peak periods.

- Storage and pricing of backup files

Backup files are stored to OBS.

- Handling exceptions in scheduled backup

If a scheduled backup task is triggered while the DCS instance is restarting or being scaled up, the scheduled backup task will be run in the next cycle.

If backing up a DCS instance fails or the backup is postponed because another task is in progress, DCS will try to back up the instance in the next cycle. A maximum of three retries are allowed within a single day.

- Retention period of backup data

Scheduled backup files are retained for up to seven days. You can configure the retention period. At the end of the retention period, most backup files of the DCS instance will be automatically deleted, but at least one backup file will be retained.

Manual backup files are retained permanently and need to be manually deleted.

Data Restoration

- Data restoration process
 - a. You can initiate a data restoration request using the DCS console.
 - b. DCS obtains the backup file from OBS.
 - c. Read/write to the DCS instance is suspended.
 - d. The original data persistence file of the master cache node is replaced by the backup file.
 - e. The new data persistence file (that is, the backup file) is reloaded.
 - f. Data is restored, and the DCS instance starts to provide read/write service again.

- Impact on service systems

Restoration tasks run on master cache nodes. During restoration, data cannot be written into or read from instances.

- Handling data restoration exceptions

If a backup file is corrupted, DCS will try to fix the backup file while restoring instance data. If the backup file is successfully fixed, the restoration proceeds. If the backup file cannot be fixed, the master/standby DCS instance will be changed back to the state in which it was before data restoration.

4.3.2 Configuring a Backup Policy

On the DCS console, you can configure an automatic backup policy. The system then backs up data in your instances according to the backup policy.

If automatic backup is not required, disable the automatic backup function in the backup policy.

Prerequisites

At least one master/standby DCS instance has been created.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner and select a region and a project.


- Step 3** In the navigation pane, choose **Cache Manager**.
- Step 4** Click the name of the DCS instance to display more details about the DCS instance.
- Step 5** On the instance details page, click **Backups & Restorations**.
- Step 6** Slide  to the right to enable automatic backup. Backup policies will be displayed.

Table 4-7 Parameters in a backup policy

Parameter	Description
Backup Schedule	Day of a week on which data in the chosen DCS instance is automatically backed up. You can select one or multiple days of a week.
Retention Period (days)	The number of days that automatically backed up data is retained. Backup data will be permanently deleted at the end of retention period and cannot be restored. Value range: 1-7.
Start Time	Time at which automatic backup starts. Value: the full hour between 00:00 to 23:00 The DCS checks backup policies once every hour. If the backup start time in a backup policy has arrived, data in the corresponding instance is backed up. NOTE Instance backup takes 5 to 30 minutes. The data added or modified during the backup process will not be backed up. To reduce the impact of backup on services, it is recommended that data should be backed up during off-peak periods. Only instances in the Running state can be backed up.

- Step 7** Click **OK**.

----End

4.3.3 Manually Backing Up a DCS Instance


You need to manually back up data in DCS instances in a timely manner. This section describes how to manually back up data in master/standby instances using the DCS console.

By default, manually backed up data is permanently retained. If backup data is no longer in use, you can delete it manually.

Prerequisites

At least one master/standby DCS instance is in the **Running** state.

Procedure

- Step 1** Log in to the DCS console.
- Step 2** Click  in the upper left corner and select a region and a project.
- Step 3** In the navigation pane, choose **Cache Manager**.
- Step 4** Click the name of the DCS instance to display more details about the DCS instance.
- Step 5** On the instance details page, click **Backups & Restorations**.
- Step 6** Click **Create Backup**.
- Step 7** In the **Create Backup** dialog box, click **OK**.

Information in the **Description** text box cannot exceed 128 bytes.

NOTE

Instance backup takes 10 to 15 minutes. The data added or modified during the backup process will not be backed up.

----End


4.3.4 Restoring a DCS Instance

On the DCS console, you can restore backup data to a chosen DCS instance.

Prerequisites

- At least one master/standby or cluster DCS instance is in the **Running** state.
- A backup task has been run to back up data in the instance to be restored and the status of the backup task is **Succeeded**.

Procedure

- Step 1** Log in to the DCS console.
- Step 2** Click  in the upper left corner and select a region and a project.
- Step 3** In the navigation pane, choose **Cache Manager**.
- Step 4** Click the name of the DCS instance to display more details about the DCS instance.
- Step 5** On the instance details page, click **Backups & Restorations**.
A list of historical backup tasks is then displayed.
- Step 6** Click **Restore** in the same row as the chosen backup task.
- Step 7** Click **OK** to start instance restoration.

Information in the **Description** text box cannot exceed 128 bytes.

The **Restoration History** tab page displays the result of the instance restoration task.

 **NOTE**

Instance restoration takes 5 to 30 minutes.

While being restored, DCS instances do not accept data operation requests from clients because existing data is being overwritten by the backup data.

----End

4.3.5 Downloading a Backup File

Due to the limitations of automatic and manual backups (automatically backed up data can be retained for a maximum of 7 days, and manually backed up data takes space in OBS), you should download the backup files and permanently save them on the local host.


This function is supported only by master/standby and cluster instances, and not by single-node instances.

Prerequisites

The instance has been backed up and the backup is still valid.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Click the name of the DCS instance to display more details about the DCS instance.

Step 5 On the instance details page, click **Backups & Restorations**.

A list of historical backup tasks is then displayed.

Step 6 Select the historical backup data to be downloaded, and click **Download**.

Step 7 In the displayed, **Download Backup File** dialog box, select either of the following two download methods.

Download methods:

- By URL
 - a. Set the URL validity period and click **Query**.
 - b. Download the backup file by using the URL list.

 **NOTE**

If you choose to copy URLs, use quotation marks to quote the URLs when running the **wget** command in Linux. For example:

```
wget 'https://obsEndpoint.com:443/redisdemo.rdb?  
parm01=value01&parm02=value02'
```

This is because the URL contains the special character and (&), which will confuse the **wget** command. Quoting the URL facilitates URL identification.

- By OBS
Perform the procedure as prompted.

----End

4.4 Migrating Data with DCS

4.4.1 Introduction to Migration with DCS

Migration Modes

DCS for Redis provides the following migration modes:

- Backup file import: The data source can be an OBS bucket or a Redis instance.
 - Importing data from an OBS bucket: Download the source Redis data and then upload it to an OBS bucket in the same region as the target DCS Redis instance. DCS will read the backup data from the OBS bucket and migrate the data into the target instance.
This migration mode can be used for migrating data from other Redis vendors or self-hosted Redis to DCS for Redis.
 - Importing data from a Redis instance: Back up the source Redis data and then migrate the backup data to DCS for Redis.
- Migrating data online: If the source and target instances are interconnected and the **SYNC** and **PSYNC** commands are supported in the source instance, data can be migrated online in full or incrementally from the source to the target.

The following table describes data migration modes supported by DCS.

 **NOTE**

Data can be migrated only from DCS Redis instances or self-hosted Redis. After data migration, change the instance connection address to the target instance address.

Data migration is not supported if the DCS instance is created by another service, such as ROMA Connect, or by calling an API.

Table 4-8 DCS data migration modes

Migration Mode	Source	Target: DCS		
		Single-Node and Master/Standby	Proxy Cluster	Redis Cluster

Importing backup files	AOF files in OBS NOTE AOF files exported from Redis 4.0/5.0 instances and other instances with RDB compression enabled cannot be imported.	√	√	×
	RDB files in OBS	√	√	√
Migrating data online	DCS for Redis: single-node or master/standby	√	√	√
	DCS for Redis: Proxy Cluster NOTE Proxy Cluster DCS Redis 3.0 instances cannot be used as the source, while Proxy Cluster DCS Redis 4.0 or 5.0 instances can.	√	√	√
	DCS for Redis: Redis Cluster	√	√	√
	Self-hosted Redis: single-node or master/standby	√	√	√
	Self-hosted Redis: proxy-based cluster	√	√	√
	Self-hosted Redis: Redis Cluster	√	√	√

	Other Redis: single-node or master/standby	×	×	×
	Other Redis: proxy-based cluster	×	×	×
	Other Redis: Redis Cluster	×	×	×

 NOTE

- **DCS for Redis** refers to Redis instances provided by DCS
- **Self-hosted Redis** refers to self-hosted Redis on the cloud, from other cloud vendors, or in on-premises data centers.
- **Other Redis** refers to Redis services provided by other cloud vendors.
- ✓: Supported. ×: Not supported.

4.4.2 Importing Backup Files

4.4.2.1 Importing Backup Files from an OBS Bucket

Scenario

Use the DCS console to migrate Redis data from Redis of other vendors or self-hosted Redis to DCS for Redis.

Simply download the source Redis data and then upload the data to an OBS bucket in the same region as the target DCS Redis instance. After you have created a migration task on the DCS console, DCS will read data from the OBS bucket and data will be migrated to the target instance.

.aof, .rbb, .zip, and .tar.gz files can be uploaded to OBS buckets. You can directly upload .aof and .rdb files or compress them into .zip or .tar.gz files before uploading.

Prerequisites

- The OBS bucket must be in the same region as the target DCS Redis instance.
- The data files to be uploaded must be in the .aof, .rdb, .zip, or .tar.gz format.
- To migrate data from a single-node or master/standby Redis instance of other cloud vendors, create a backup task and download the backup file.
- To migrate data from a cluster Redis instance of other cloud vendors, download all backup files and upload all of them to the OBS bucket. Each backup file contains data for a shard of the instance.
- .rdb backup files of self-hosted Redis 5.0 cannot be imported. .rdb backup files of self-hosted Redis 3.0 or 4.0 can be exported using redis-cli. .rdb files of

other cloud Redis can be exported only by creating backup tasks, and cannot be exported by running commands in redis-cli.

- Redis Cluster instances only support .rdb files and do not support .aof files.

Step 1: Prepare the Target DCS Redis Instance

- If a DCS Redis instance is not available, create one first. For details, see [Creating a DCS Redis Instance](#).
- If a DCS Redis instance is available, you do not need to create a new one. However, you must clear the instance data before the migration.
 - If the target instance is Redis 4.0 or 5.0, clear the data by referring to [Clearing DCS Instance Data](#).
 - If the target instance is a DCS Redis 3.0 instance, run the **FLUSHALL** command to clear data.

You can use a DCS Redis 3.0, 4.0, or 5.0 instance as the target instance.

Step 2: Create an OBS Bucket and Upload Backup Files

Step 1 Create an OBS bucket.

1. Log in to the OBS Console and click **Create Bucket**.
2. Select a region.
The OBS bucket must be in the same region as the target DCS Redis instance.
3. Specify **Bucket Name**.
The bucket name must meet the naming rules specified on the console.
4. Set **Storage Class** to **Standard**, **Warm** or **Cold**.
5. Set **Bucket Policy** to **Private**, **Public Read**, or **Public Read and Write**.
6. Configure default encryption.
7. Click **Create Now**.

Step 2 Upload the backup data files to the OBS bucket by using OBS Browser+.

If the backup file to be uploaded does not exceed 5 GB, upload the file using the OBS console by referring to step [3](#).

If the backup file to be uploaded is larger than 5 GB, perform the following steps to upload the file using OBS Browser+.

1. Download OBS Browser+.
For details, see section "Downloading OBS Browser+" in *Object Storage Service (OBS) Tools Guide (OBS Browser+)*.
2. Install OBS Browser+.
For details, see section "Installing OBS Browser+" in *Object Storage Service (OBS) Tools Guide (OBS Browser+)*.
3. Log in to OBS Browser+.
For details, see section "Logging In to OBS Browser+" in *Object Storage Service (OBS) Tools Guide (OBS Browser+)*.
4. Creates a bucket.

5. Upload backup data.

Step 3 On the OBS console, upload the backup data files to the OBS bucket.

Perform the following steps if the backup file size does not exceed 5 GB:

1. In the bucket list, click the name of the created bucket.
2. In the navigation pane, choose **Objects**.
3. On the **Objects** tab page, click **Upload Object**.
4. A maximum of 100 files can be uploaded at a time. The total size cannot exceed 5 GB.


To upload objects, drag files or folders to the **Upload Object** area or click **add file**.

5. Click **Upload**.

----End

Step 3: Create a Migration Task

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Data Migration**.

Step 4 Click **Create Backup Import Task**.

Step 5 Specify **Task Name** and **Description**.

Step 6 Select **OBS Bucket** as the data source and then select the OBS bucket to which you have uploaded backup files.

 **NOTE**

You can upload files in the .aof, .rdb, .zip, or .tar.gz format.

Step 7 Select the backup files whose data is to be migrated.

Step 8 Select the target DCS Redis instance prepared in [Step 1: Prepare the Target DCS Redis Instance](#).

Step 9 Enter the password of the target instance. Click **Test Connection** to verify the password.

Step 10 Click **Next**.

Step 11 Confirm the migration task details and click **Submit**.

Go back to the data migration task list. After the migration is successful, the task status changes to **Successful**.

----End

4.4.2.2 Importing Backup Files from Redis

Scenario

Use the DCS console to migrate Redis data from self-hosted Redis to DCS for Redis.

Simply back up your Redis data, create a migration task on the DCS console, and then import the backup to a DCS Redis instance.

Prerequisites

A master/standby or cluster DCS Redis instance has been created as the target for the migration. The source instance has data and has been backed up.

Step 1: Obtain the Source Instance Name and Password

Obtain the name of the source Redis instance.

Step 2: Prepare the Target DCS Redis Instance

- If a DCS Redis instance is not available, create one first. For details, see [Creating a DCS Redis Instance](#).
- If a DCS Redis instance is available, you do not need to create a new one. However, you must clear the instance data before the migration.
 - If the target instance is Redis 4.0 or 5.0, clear the data by referring to [Clearing DCS Instance Data](#).
 - If the target instance is a DCS Redis 3.0 instance, run the **FLUSHALL** command to clear data.

You can use a DCS Redis 3.0, 4.0, or 5.0 instance as the target instance.

Step 3: Create a Migration Task

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Data Migration**.

Step 4 Click **Create Backup Import Task**.

Step 5 Enter the task name and description.

Step 6 Set **Data Source** to **Redis**.

Step 7 For source Redis, select the instance prepared in [Step 1: Obtain the Source Instance Name and Password](#).

Step 8 Select the backup task whose data is to be migrated.

Step 9 Select the target instance created in [Step 2: Prepare the Target DCS Redis Instance](#).

Step 10 Enter the password of the target instance. Click **Test Connection** to verify the password.

Step 11 Click **Next**.

Step 12 Confirm the migration task details and click **Submit**.

Go back to the data migration task list. After the migration is successful, the task status changes to **Successful**.

----End

4.4.3 Migrating Data Online

Scenario

If the source and target instances are interconnected and the **SYNC** and **PSYNC** commands are supported in the source instance, data can be migrated online in full or incrementally from the source to the target.

Prerequisites

- Before migrating data, read through [Introduction to Migration with DCS](#) to learn about the DCS data migration function and select an appropriate target instance.
- To migrate data from a single-node or master/standby instance to a Redis Cluster instance, check if any data exists in DBs other than DB0 in the source instance. If yes, move the data to DB0 by using the open-source tool Rump. Otherwise, the migration will fail because a Redis Cluster instance has only one DB. For details about the migration operations, see [Online Migration with Rump](#).

Obtaining Information About the Source Redis Instance

- If the source is a cloud Redis instance, obtain its name.
- If the source is self-hosted Redis, obtain its IP address or domain name and port number.

Prepare the Target DCS Redis Instance

- If a target DCS Redis instance is not available, create one first. For details, see [Creating a DCS Redis Instance](#).
- If a target instance is available, you do not need to create a new one. However, you must clear the instance data before the migration. For details, see [Clearing DCS Instance Data](#).

Requirements on the Network Between the Online Migration Task, Source Redis, and Target Redis

NOTE


- If the source or target of online migration is **Redis in the cloud**, the selected Redis instance must be in the same VPC as the migration task. Otherwise, the migration task may fail to connect to the cloud Redis instance.
- In special scenarios, if you have enabled cross-VPC access between the migration task and the cloud Redis instance, the cloud Redis instance and the migration task can be in different VPCs.

Table 4-9 lists the requirements on the network between the online migration task, source Redis, and target Redis.

Table 4-9 Requirements on the network between the online migration task, source Redis, and target Redis

Source Redis Type	Target Redis Type	Network Requirement on Online Migration
Redis in the cloud	Redis in the cloud	When creating an online migration task, ensure that the online migration task is in the same VPC as the source and target Redis. If they are not in the same VPC, enable cross-network access between the migration task and the source and target Redis. To enable cross-network access, create a VPC peering connection by referring to section "VPC Peering Connection" in <i>VPC User Guide</i> .
Redis in the cloud	Self-hosted Redis	When creating an online migration task, ensure that the migration task and the source Redis are in the same VPC. Then, enable cross-network access between the migration task and the target Redis. To enable cross-network access, create a VPC peering connection by referring to section "VPC Peering Connection" in <i>VPC User Guide</i> .
Self-hosted Redis	Redis in the cloud	When creating an online migration task, ensure that the migration task and the target Redis are in the same VPC. Then, enable cross-network access between the migration task and the source Redis. To enable cross-network access, create a VPC peering connection by referring to section "VPC Peering Connection" in <i>VPC User Guide</i> .
Self-hosted Redis	Self-hosted Redis	After creating an online migration task, enable cross-network access between the migration task and the source and target Redis, respectively. To enable cross-network access, create a VPC peering connection by referring to section "VPC Peering Connection" in <i>VPC User Guide</i> .

Create a Migration Task

- Step 1** Log in to the DCS console.
- Step 2** Click  in the upper left corner and select a region and a project.
- Step 3** In the navigation pane, choose **Data Migration**. The migration task list is displayed.

Step 4 Click **Create Online Migration Task**.

Step 5 Enter the task name and description.

Step 6 Select a VPC and security group.

Step 7 Click **Next**.

Step 8 Click **Submit**.

----End

Configuring the Online Migration Task

Step 1 On the **Online Migration** tab page, click **Configure** in the row containing the online migration task you just created.

Step 2 Specify **Migration Type**.

Supported migration types are **Full** and **Full + incremental**, which are described in [Table 4-10](#).

Table 4-10 Migration type description

Migration Type	Description
Full	Suitable for scenarios where services can be interrupted. Data is migrated at one time. Source instance data updated during the migration will not be migrated to the target instance.
Full + incremental	Suitable for scenarios requiring minimal service downtime. The incremental migration parses logs to ensure data consistency between the source and target instances. Once incremental migration starts, it remains Migrating until you click Stop in the Operation column. After the migration is stopped, data in the source instance will not be lost, but data will not be written to the target instance. When the transmission network is stable, the delay of incremental migration is within seconds. The actual delay depends on the transmission quality of the network link.

Step 3 Configure source Redis and target Redis.

- Source Redis Type:** Select **Redis in the cloud** or **Self-hosted Redis** as required.
 - Redis in the cloud:** a DCS Redis instance that is in the same VPC as the migration task
 - Self-hosted Redis:** self-hosted Redis in another cloud, or in on-premises data centers. If you select this option, enter Redis addresses.
- If the instance is password-protected, you can click **Test Connection** to check whether the instance password is correct and whether the network is connected.

Step 4 For **Target Instance**, select the DCS Redis Instance prepared in [Prepare the Target DCS Redis Instance](#).

If the instance is password-protected, you can click **Test Connection** to check whether the instance password meets the requirements.

 **NOTE**

If the source and target Redis instances are connected but are in different regions of DCS, you can only select **Self-hosted Redis** for **Target Redis Type** and enter the instance addresses, regardless of whether the target Redis instance is self-hosted or in the cloud.

Step 5 Click **Next**.

Step 6 Confirm the migration task details and click **Submit**.

Go back to the data migration task list. After the migration is successful, the task status changes to **Successful**.

 **NOTE**

If the migration type is full+incremental, the migration task status will remain **Migrating** until you click **Stop**.

----End

4.5 Managing Passwords

4.5.1 DCS Instance Passwords

Passwords can be configured to control access to your DCS instances, ensuring the security of your data.

 **NOTE**

After 5 consecutive incorrect password attempts, the account for accessing the chosen DCS instance will be locked for 5 minutes. Passwords cannot be changed during the lockout period.

The password must meet the following requirements:

- Cannot be left blank.
- Cannot be the same as the old password.
- Can contain 8 to 32 characters.
- Must contain at least three of the following character types:
 - Lowercase letters
 - Uppercase letters
 - Digits
 - special characters (`~!@#$%^&*()-_+=\|{};<.>/?`)

Using Passwords Securely

1. Hide the password when using redis-cli.

If the **-a <password>** option is used in redis-cli in Linux, the password is prone to leakage because it is logged and kept in the history. You are advised not to

use **-a <password>** when running commands in redis-cli. After connecting to Redis, run the **auth** command to complete authentication as shown in the following example:

```
$ redis-cli -h 192.168.0.148 -p 6379
redis 192.168.0.148:6379>auth yourPassword
OK
redis 192.168.0.148:6379>
```

2. Use interactive password authentication or switch between users with different permissions.

If the script involves DCS instance access, use interactive password authentication. To enable automatic script execution, manage the script as another user and authorize execution using `sudo`.

3. Use an encryption module in your application to encrypt the password.

4.5.2 Changing Instance Passwords

On the DCS console, you can change the password required for accessing your DCS instance.


NOTE

- You cannot change the password of a DCS instance in password-free mode.
- The DCS instance for which you want to change the password is in the **Running** state.
- The new password takes effect immediately on the server without requiring a restart. The client must reconnect to the server using the new password after a `pconnect` connection is closed. (The old password can still be used before disconnection.)

Prerequisites

At least one DCS instance has been created.

Procedure

- Step 1** Log in to the DCS console.
- Step 2** Click  in the upper left corner and select a region and a project.
- Step 3** In the navigation pane, choose **Cache Manager**.
- Step 4** Choose **More > Change Password** in the same row as the chosen instance.
- Step 5** In the displayed dialog box, set **Old Password**, **New Password**, and **Confirm Password**.

 **NOTE**

After 5 consecutive incorrect password attempts, the account for accessing the chosen DCS instance will be locked for 5 minutes. Passwords cannot be changed during the lockout period.

The password must meet the following requirements:

- Cannot be left blank.
- The new password cannot be the same as the old password.
- Can contain 8 to 32 characters.
- Must contain at least three of the following character types:
 - Lowercase letters
 - Uppercase letters
 - Digits
 - special characters (^~!@#\$\$%^&*()-_+=+\\|{};<.>/?)

Step 6 In the **Change Password** dialog box, click **OK** to confirm the password change.

----End

4.5.3 Resetting Instance Passwords

On the DCS console, you can configure a new password if you forget your instance password.

 **NOTE**

- For a DCS Redis or Memcached instance, you can change it from password mode to password-free mode or from password-free mode to password mode by resetting its password. For details, see [Changing Password Settings for DCS Redis Instances](#) and [Changing Password Settings for DCS Memcached Instances](#).
- The DCS instance for which you want to reset the password is in the **Running** state.

Prerequisites

At least one DCS instance has been created.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 Choose **More > Reset Password** in the same row as the chosen instance.

Step 5 In the **Reset Password** dialog box, enter a new password and confirm the password.

 **NOTE**

The password must meet the following requirements:

- Cannot be left blank.
- Can contain 8 to 32 characters.
- Contain at least three of the following character types:
 - Lowercase letters
 - Uppercase letters
 - Digits
 - special characters (^~!@#\$%^&*()-_+=\|{};,<.>/?)

Step 6 Click **OK**.

 **NOTE**

The system will display a success message only after the password is successfully reset on all nodes. If the reset fails, the instance will restart and the password of the cache instance will be restored.

----End

4.5.4 Changing Password Settings for DCS Redis Instances

Scenario

DCS Redis instances can be accessed with or without passwords. After an instance is created, you can change its password setting in the following scenarios:

- To enable public access for a password-free DCS Redis instance, you must change the instance to password-protected mode before enabling public access.
- To access a DCS Redis instance in password-free mode, you can enable password-free access to clear the existing password of the instance.

 **NOTE**

- To change the password setting, the DCS Redis instance must be in the **Running** state.
- Password-free access may compromise security. You can set a password by using the password reset function.
- For security purposes, password-free access must be disabled when public access is enabled.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 To change the password setting for a DCS Redis instance, choose **Operation > More > Reset Password** in the same row as the chosen instance.

Step 5 In the **Reset Password** dialogue box, perform either of the following operations as required:

- From password-protected to password-free:
Switch the toggle for **Password-Free Access** and click **OK**.
- From password-free to password-protected:
Enter a password, confirm the password, and click **OK**.

----End

4.5.5 Changing Password Settings for DCS Memcached Instances


Scenario

DCS Memcached instances can be accessed with or without passwords. After an instance is created, you can change its password setting in the following scenarios:

- If you want to access a password-protected DCS Memcached instance without the username and password, you can enable password-free access to clear the username and password of the instance.
The Memcached text protocol does not support username and password authentication. To access a DCS Memcached instance by using the Memcached text protocol, you must enable password-free access to the instance.
- If you want to access a password-free DCS Memcached instance using a username and password, you can set a password for the instance using the password reset function.

Procedure

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 To enable password-free access to a DCS Memcached instance, choose **Operation** > **More** > **Reset Password** in the same row as the chosen instance.

Step 5 In the **Reset Password** dialogue box, perform either of the following operations as required:

- From password-protected to password-free:
Switch the toggle for **Password-Free Access** and click **OK**.
- From password-free to password-protected:
Enter a password, confirm the password, and click **OK**.

----End

5 Monitoring

5.1 DCS Metrics

Introduction

This section describes DCS metrics reported to Cloud Eye as well as their namespaces and dimensions. You can use the Cloud Eye console or call APIs to query the DCS metrics and alarms.

Different types of instances are monitored on different dimensions.

- **Single-node:**
Single-node instances are monitored on the instance dimension. The monitoring is conducted on the Redis Server.
- **Master/standby:**
Master/Standby instances are monitored on the instance and Redis Server dimensions. Instance monitoring covers the master node, while Redis Server monitoring covers the master and standby nodes.
- **Cluster:**
Proxy Cluster instances are monitored on the instance, Redis Server, and proxy dimensions. Instance monitoring covers the aggregated master node data, Redis Server monitoring covers each shard in the cluster, and proxy monitoring covers each proxy in the cluster.
Redis Cluster instances are monitored on the instance and Redis Server dimensions. Instance monitoring covers the aggregated master node data and Redis Server monitoring covers each shard in the cluster.

Namespace

SYS.DCS

DCS Redis 3.0 Instance Metrics

 NOTE

The **Monitored Objects and Dimensions** column lists instances and dimensions that support the corresponding metrics.

Table 5-1 DCS Redis 3.0 instance metrics

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
cpu_usage	CPU Usage	The monitored object's maximum CPU usage among multiple sampling values in a monitoring period Unit: %	0–100%	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
memory_usage	Memory Usage	Memory consumed by the monitored object Unit: %	0–100%	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
net_in_throughput	Network Input Throughput	Inbound throughput per second on a port Unit: byte/s	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
net_out_throughput	Network Output Throughput	Outbound throughput per second on a port Unit: byte/s	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
connected_clients	Connected Clients	Number of connected clients (excluding those from slave nodes)	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
client_longest_out_list	Client Longest Output List	Longest output list among current client connections	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
client_biggest_in_buf	Client Biggest Input Buf	Maximum input data length among current client connections Unit: byte	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
blocked_clients	Blocked Clients	Number of clients suspended by block operations such as BLPOP, BRPOP, and BRPOPLPUSH	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
used_memory	Used Memory	Number of bytes used by the Redis server Unit: byte	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
used_memory_rss	Used Memory RSS	Resident set size (RSS) memory that the Redis server has used, which is the memory that actually resides in the memory, including all stack and heap memory but not swapped-out memory Unit: byte	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
used_memory_peak	Used Memory Peak	Peak memory consumed by Redis since the Redis server last started Unit: byte	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
used_memory_lua	Used Memory Lua	Number of bytes used by the Lua engine Unit: byte	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
memory_fragmentation_ratio	Memory Fragmentation Ratio	Current memory fragmentation, which is the ratio between used_memory_rss/used_memory .	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
total_connections_received	New Connections	Number of connections received during the monitoring period	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
total_commands_processed	Commands Processed	Number of commands processed during the monitoring period	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
instantaneous_ops	Ops per Second	Number of commands processed per second	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
total_net_input_bytes	Network Input Bytes	Number of bytes received during the monitoring period Unit: byte	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
total_net_output_bytes	Network Output Bytes	Number of bytes sent during the monitoring period Unit: byte	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
instantaneous_input_kbps	Input Flow	Instantaneous input traffic Unit: kbit/s	≥ 0 kbits/s	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
instantaneous_output_kbps	Output Flow	Instantaneous output traffic Unit: kbit/s	≥ 0 kbits/s	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
rejected_connections	Rejected Connections	Number of connections that have exceeded maxclients and been rejected during the monitoring period	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
expired_keys	Expired Keys	Number of keys that have expired and been deleted during the monitoring period	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
evicted_keys	Evicted Keys	Number of keys that have been evicted and deleted during the monitoring period	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
keyspace_hits	Keyspace Hits	Number of successful lookups of keys in the main dictionary during the monitoring period	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
keyspace_misses	Keyspace Misses	Number of failed lookups of keys in the main dictionary during the monitoring period	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
pubsub_channels	PubSub Channels	Number of Pub/Sub channels	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
pubsub_patterns	PubSub Patterns	Number of Pub/Sub patterns	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
keyspace_hits_perc	Hit Rate	Ratio of the number of Redis cache hits to the number of lookups. Calculation: keyspace_hits / (keyspace_hits + keyspace_misses) Unit: %	0–100%	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
command_max_delay	Maximum Command Latency	Maximum latency of commands Unit: ms	≥ 0 ms	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
auth_errors	Authentication Failures	Number of failed authentications	≥ 0	Monitored object: Single-node or master/standby DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
is_slow_log_exist	Slow Query Logs	Existence of slow query logs in the instance	<ul style="list-style-type: none"> • 1: yes • 0: no 	Monitored object: Single-node or master/standby DCS Redis instance Dimension: dcs_instance_id	1 minute
keys	Keys	Number of keys in Redis	≥ 0	Monitored object: Single-node or master/standby DCS Redis instance Dimension: dcs_instance_id	1 minute

DCS Redis 4.0 and 5.0 Instance Metrics

 NOTE

The **Monitored Objects and Dimensions** column lists instances and dimensions that support the corresponding metrics.

Table 5-2 DCS Redis 4.0 and 5.0 instance metrics

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
cpu_usage	CPU Usage	The monitored object's maximum CPU usage among multiple sampling values in a monitoring period Unit: %	0–100%	Monitored object: Single-node or master/standby DCS Redis instance Dimension: dcs_instance_id	1 minute
command_max_delay	Maximum Command Latency	Maximum latency of commands Unit: ms	≥ 0 ms	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
total_connections_received	New Connections	Number of connections received during the monitoring period	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
is_slow_log_exist	Slow Query Logs	Existence of slow query logs in the instance	<ul style="list-style-type: none"> • 1: yes • 0: no 	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
memory_usage	Memory Usage	Memory consumed by the monitored object Unit: %	0–100%	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
expires	Keys With an Expiration	Number of keys with an expiration in Redis	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
keyspace_hits_perc	Hit Rate	Ratio of the number of Redis cache hits to the number of lookups. Calculation: keyspace_hits / (keyspace_hits + keyspace_misses) Unit: %	0–100%	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
used_memory	Used Memory	Number of bytes used by the Redis server Unit: byte	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
used_memory_dataset	Used Memory Dataset	Dataset memory that the Redis server has used Unit: byte	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
used_memory_dataset_percent	Used Memory Dataset Ratio	Percentage of dataset memory that the Redis server has used Unit: %	0–100%	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
used_memory_rss	Used Memory RSS	Resident set size (RSS) memory that the Redis server has used, which is the memory that actually resides in the memory, including all stack and heap memory but not swapped-out memory Unit: byte	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
instantaneous_ops	Ops per Second	Number of commands processed per second	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
keyspace_misses	Keyspace Misses	Number of failed lookups of keys in the main dictionary during the monitoring period	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
keys	Keys	Number of keys in Redis	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
blocked_clients	Blocked Clients	Number of clients suspended by block operations	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
connected_clients	Connected Clients	Number of connected clients (excluding those from slave nodes)	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
del	DEL	Number of DEL commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
evicted_keys	Evicted Keys	Number of keys that have been evicted and deleted during the monitoring period	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
expire	EXPIRE	Number of EXPIRE commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
expired_keys	Expired Keys	Number of keys that have expired and been deleted during the monitoring period	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
get	GET	Number of GET commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
hdel	HDEL	Number of HDEL commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
hget	HGET	Number of HGET commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
hmget	HMGET	Number of HMGET commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
hmset	HMSET	Number of HMSET commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
hset	HSET	Number of HSET commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
instantaneous_input_kbps	Input Flow	Instantaneous input traffic Unit: KB/s	≥ 0 KB/s	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
instantaneous_output_kbps	Output Flow	Instantaneous output traffic Unit: KB/s	≥ 0 KB/s	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
memory_frag_ratio	Memory Fragmentation Ratio	Ratio between Used Memory RSS and Used Memory	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
mget	MGET	Number of MGET commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
mset	MSET	Number of MSET commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
pubsub_channels	PubSub Channels	Number of Pub/Sub channels	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
pubsub_patterns	PubSub Patterns	Number of Pub/Sub patterns	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
set	SET	Number of SET commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
used_memory_lua	Used Memory Lua	Number of bytes used by the Lua engine Unit: byte	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
used_memory_peak	Used Memory Peak	Peak memory consumed by Redis since the Redis server last started Unit: byte	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
sadd	Sadd	Number of SADD commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
smembers	Smembers	Number of SMEMBERS commands processed per second Unit: Count/s	0–500,000	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
rx_controlled	Flow Control Times	Number of flow control times during the monitoring period Unit: Count	≥ 0	Monitored object: Redis Cluster instance Dimension: dcs_instance_id	1 minute
bandwidth_usage	Bandwidth Usage	Percentage of the used bandwidth to the maximum bandwidth limit	0–200%	Monitored object: Redis Cluster instance Dimension: dcs_instance_id	1 minute
keyspace_misses	Keyspace Misses	Number of failed lookups of keys in the main dictionary during the monitoring period	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
used_memory_dataset	Used Memory Dataset	Dataset memory that the Redis server has used	≥ 0	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute
used_memory_dataset_percent	Used Memory Dataset Ratio	Percentage of dataset memory that server has used	0–100%	Monitored object: Single-node, master/standby, or cluster DCS Redis instance Dimension: dcs_instance_id	1 minute

Node Metrics of DCS Redis Instances

 NOTE

- The following describes the metrics for cluster DCS instances. For Proxy Cluster DCS Redis 3.0 instances, the monitoring covers Redis Servers and Proxies. For Redis Cluster DCS Redis 4.0 and 5.0 instances, the monitoring only covers Redis Servers. For details, see [Table 5-3](#) and [Table 5-4](#).
- The **Monitored Objects and Dimensions** column lists instances and dimensions that support the corresponding metrics.

Table 5-3 Redis Server metrics of DCS instances

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
cpu_usage	CPU Usage	The monitored object's maximum CPU usage among multiple sampling values in a monitoring period Unit: %	0–100%	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
memory_usage	Memory Usage	Memory consumed by the monitored object Unit: %	0–100%	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_cluster_redis_node	1 minute
connected_clients	Connected Clients	Number of connected clients (excluding those from slave nodes)	≥ 0	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
client_longest_out_list	Client Longest Output List	Longest output list among current client connections	≥ 0	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
client_biggest_in_buf	Client Biggest Input Buf	Maximum input data length among current client connections Unit: byte	≥ 0	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
blocked_clients	Blocked Clients	Number of clients suspended by block operations such as BLPOP, BRPOP, and BRPOPLPUSH	≥ 0	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
used_memory	Used Memory	Number of bytes used by the Redis server Unit: byte	≥ 0	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
used_memory_rss	Used Memory RSS	RSS memory that the Redis server has used, which including all stack and heap memory but not swapped-out memory Unit: byte	≥ 0	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
used_memory_peak	Used Memory Peak	Peak memory consumed by Redis since the Redis server last started Unit: byte	≥ 0	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
used_memory_lua	Used Memory Lua	Number of bytes used by the Lua engine Unit: byte	≥ 0	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
memory_frag_ratio	Memory Fragmentation Ratio	Current memory fragmentation, which is the ratio between used_memory_rss/used_memory .	≥ 0	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
total_connections_received	New Connections	Number of connections received during the monitoring period	≥ 0	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
total_commands_processed	Commands Processed	Number of commands processed during the monitoring period	≥ 0	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
instantaneous_ops	Ops per Second	Number of commands processed per second	≥ 0	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
total_net_input_bytes	Network Input Bytes	Number of bytes received during the monitoring period Unit: byte	≥ 0	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
total_net_output_bytes	Network Output Bytes	Number of bytes sent during the monitoring period Unit: byte	≥ 0	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
instantaneous_input_kbps	Input Flow	Instantaneous input traffic Unit: KB/s	≥ 0 KB/s	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
instantaneous_output_kbps	Output Flow	Instantaneous output traffic Unit: KB/s	≥ 0 KB/s	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
rejected_connections	Rejected Connections	Number of connections that have exceeded maxclients and been rejected during the monitoring period	≥ 0	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
expired_keys	Expired Keys	Number of keys that have expired and been deleted during the monitoring period	≥ 0	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
evicted_keys	Evicted Keys	Number of keys that have been evicted and deleted during the monitoring period	≥ 0	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
pubsub_channels	PubSub Channels	Number of Pub/Sub channels	≥ 0	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
pubsub_patterns	PubSub Patterns	Number of Pub/Sub patterns	≥ 0	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
keyspace_hits_perc	Hit Rate	Ratio of the number of Redis cache hits to the number of lookups. Calculation: keyspace_hits / (keyspace_hits + keyspace_misses) Unit: %	0–100%	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
command_max_delay	Maximum Command Latency	Maximum latency of commands Unit: ms	≥ 0 ms	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
is_slow_log_exist	Slow Query Logs	Existence of slow query logs in the node	<ul style="list-style-type: none"> • 1: yes • 0: no 	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
keys	Keys	Number of keys in Redis	≥ 0	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
sadd	Sadd	Number of SADD commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of a cluster DCS Redis 3.0, 4.0, or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
smembers	Smembers	Number of SMEMBERS commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of a cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
ms_repl_offset	Replication Gap	Data synchronization gap between the master and the replica	-	Monitored object: Replica of a cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
del	DEL	Number of DEL commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of a cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
expire	EXPIRE	Number of EXPIRE commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of a cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
get	GET	Number of GET commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of a cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
hdel	HDEL	Number of HDEL commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of a cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
hget	HGET	Number of HGET commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of a cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
hmget	HMGET	Number of HMGET commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of a cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
hmset	HMSET	Number of HMSET commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of a cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
hset	HSET	Number of HSET commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of a cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
mget	MGET	Number of MGET commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of a cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
mset	MSET	Number of MSET commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of a cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
set	SET	Number of SET commands processed per second Unit: Count/s	0–500,000	Monitored object: Redis Server of a cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute
rx_controlled	Flow Control Times	Number of flow control times during the monitoring period Unit: Count	≥ 0	Monitored object: Redis Server of a cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
bandwidth_usage	Bandwidth Usage	Percentage of the used bandwidth to the maximum bandwidth limit	0–200%	Monitored object: Redis Server of a cluster DCS Redis 4.0 or 5.0 instance Dimension: dcs_instance_id dcs_cluster_redis_node	1 minute

Table 5-4 Proxy metrics

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
cpu_usage	CPU Usage	The monitored object's maximum CPU usage among multiple sampling values in a monitoring period Unit: %	0–100%	Monitored object: Proxy in a Proxy Cluster DCS Redis 3.0 instance Dimension: dcs_instance_id dcs_cluster_proxy_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
memory_usage	Memory Usage	Memory consumed by the monitored object Unit: %	0-100%	Monitored object: Proxy in a Proxy Cluster DCS Redis 3.0 instance Dimension: dcs_instance_id dcs_cluster_proxy_node	1 minute
p_connected_clients	Connected Clients	Number of connected clients	≥ 0	Monitored object: Proxy in a Proxy Cluster DCS Redis 3.0 instance Dimension: dcs_instance_id dcs_cluster_proxy_node	1 minute
max_rxpck_per_sec	Max. NIC Data Packet Receive Rate	Maximum number of data packets received by the proxy NIC per second during the monitoring period Unit: packages/second	0-10,000,000	Monitored object: Proxy in a Proxy Cluster DCS Redis 3.0 instance Dimension: dcs_instance_id dcs_cluster_proxy_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
max_txpck_per_sec	Max. NIC Data Packet Transmit Rate	Maximum number of data packets transmitted by the proxy NIC per second during the monitoring period Unit: packages/second	0–10,000,000	Monitored object: Proxy in a Proxy Cluster DCS Redis 3.0 instance Dimension: dcs_instance_id dcs_cluster_proxy_node	1 minute
max_rxkB_per_sec	Maximum Inbound Bandwidth	Largest volume of data received by the proxy NIC per second Unit: KB/s	≥ 0 KB/s	Monitored object: Proxy in a Proxy Cluster DCS Redis 3.0 instance Dimension: dcs_instance_id dcs_cluster_proxy_node	1 minute
max_txkB_per_sec	Maximum Outbound Bandwidth	Largest volume of data transmitted by the proxy NIC per second Unit: KB/s	≥ 0 KB/s	Monitored object: Proxy in a Proxy Cluster DCS Redis 3.0 instance Dimension: dcs_instance_id dcs_cluster_proxy_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
avg_rxpck_per_sec	Average NIC Data Packet Receive Rate	Average number of data packets received by the proxy NIC per second during the monitoring period Unit: packages/second	0–10,000,000	Monitored object: Proxy in a Proxy Cluster DCS Redis 3.0 instance Dimension: dcs_instance_id dcs_cluster_proxy_node	1 minute
avg_txpck_per_sec	Average NIC Data Packet Transmit Rate	Average number of data packets transmitted by the proxy NIC per second during the monitoring period Unit: packages/second	0–10,000,000	Monitored object: Proxy in a Proxy Cluster DCS Redis 3.0 instance Dimension: dcs_instance_id dcs_cluster_proxy_node	1 minute
avg_rxkB_per_sec	Average Inbound Bandwidth	Average volume of data received by the proxy NIC per second Unit: KB/s	≥ 0 KB/s	Monitored object: Proxy in a Proxy Cluster DCS Redis 3.0 instance Dimension: dcs_instance_id dcs_cluster_proxy_node	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
avg_txB_per_sec	Average Outbound Bandwidth	Average volume of data transmitted by the proxy NIC per second Unit: KB/s	≥ 0 KB/s	Monitored object: Proxy in a Proxy Cluster DCS Redis 3.0 instance Dimension: dcs_instance_id dcs_cluster_proxy_node	1 minute

DCS Memcached Instance Metrics

Table 5-5 DCS Memcached instance metrics

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
cpu_usage	CPU Usage	The monitored object's maximum CPU usage among multiple sampling values in a monitoring period Unit: %	0-100%	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
memory_usage	Memory Usage	Memory consumed by the monitored object Unit: %	0–100%	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
net_in_throughput	Network Input Throughput	Inbound throughput per second on a port Unit: byte/s	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
net_out_throughput	Network Output Throughput	Outbound throughput per second on a port Unit: byte/s	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_connected_clients	Connected Clients	Number of connected clients (excluding those from slave nodes)	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
mc_used_memory	Used Memory	Number of bytes used by Memcached Unit: byte	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_used_memory_rss	Used Memory RSS	RSS memory used is the memory that actually resides in the memory, including all stack and heap memory but not swapped-out memory Unit: byte	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_used_memory_peak	Used Memory Peak	Peak memory consumed since the server last started Unit: byte	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_memory_fragmentation_ratio	Memory Fragmentation Ratio	Current memory fragmentation, which is the ratio between used_memory_rss/used_memory .	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
mc_connections_received	New Connections	Number of connections received during the monitoring period	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_commands_processed	Commands Processed	Number of commands processed during the monitoring period	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_instantaneous_ops	Ops per Second	Number of commands processed per second	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_net_input_bytes	Network Input Bytes	Number of bytes received during the monitoring period Unit: byte	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
mc_net_output_bytes	Network Output Bytes	Number of bytes sent during the monitoring period Unit: byte	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_instantaneous_input_kbps	Input Flow	Instantaneous input traffic Unit: KB/s	≥ 0 KB/s	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_instantaneous_output_kbps	Output Flow	Instantaneous output traffic Unit: KB/s	≥ 0 KB/s	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_rejected_connections	Rejected Connections	Number of connections that have exceeded maxclients and been rejected during the monitoring period	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
mc_expired_keys	Expired Keys	Number of keys that have expired and been deleted during the monitoring period	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_evicted_keys	Evicted Keys	Number of keys that have been evicted and deleted during the monitoring period	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_cmd_get	Number of Retrieval Requests	Number of received data retrieval requests	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_cmd_set	Number of Storage Requests	Number of received data storage requests	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
mc_cmd_flush	Number of Flush Requests	Number of received data clearance requests	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_cmd_touch	Number of Touch Requests	Number of received requests for modifying the validity period of data	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_get_hits	Number of Retrieval Hits	Number of successful data retrieval operations	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_get_misses	Number of Retrieval Misses	Number of failed data retrieval operations due to key nonexistence	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
mc_delete_hits	Number of Delete Hits	Number of successful data deletion operations	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_delete_misses	Number of Delete Misses	Number of failed data deletion operations due to key nonexistence	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_incr_hits	Number of Increment Hits	Number of successful increment operations	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_incr_misses	Number of Increment Misses	Number of failed increment operations due to key nonexistence	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
mc_decr_hits	Number of Decrement Hits	Number of successful decrement operations	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_decr_misses	Number of Decrement Misses	Number of failed decrement operations due to key nonexistence	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_cas_hits	Number of CAS Hits	Number of successful CAS operations	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_cas_misses	Number of CAS Misses	Number of failed CAS operations due to key nonexistence	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
mc_cas_ba dval	Number of CAS Values Not Matched	Number of failed CAS operations due to CAS value mismatch	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_touch_hits	Number of Touch Hits	Number of successful requests for modifying the validity period of data	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_touch_misses	Number of Touch Misses	Number of failed requests for modifying the validity period of data due to key nonexistence	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_auth_c mds	Authentication Requests	Number of authentication requests	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
mc_auth_errors	Authentication Failures	Number of failed authentication requests	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_curr_items	Number of Items Stored	Number of stored data items	≥ 0	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_command_max_delay	Maximum Command Latency	Maximum latency of commands Unit: ms	≥ 0 ms	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute
mc_is_slow_log_exist	Slow Query Logs	Existence of slow query logs in the instance	<ul style="list-style-type: none"> • 1: yes • 0: no 	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute

Metric ID	Metric	Description	Value Range	Monitored Object and Dimension	Monitoring Period (Raw Data)
mc_keyspace_hits_perc	Hit Rate	Ratio of the number of Memcached cache hits to the number of lookups Unit: %	0-100%	Monitored object: DCS Memcached instance Dimension: dcs_memcached_instance_id	1 minute


Dimensions

Key	Value
dcs_instance_id	DCS Redis instance
dcs_cluster_redis_node	Redis Server
dcs_cluster_proxy_node	Proxy
dcs_memcached_instance_id	DCS Memcached instance

5.2 Viewing DCS Monitoring Metrics

You can view DCS instance metrics on the **Performance Monitoring** page.

Procedure

- Step 1** Log in to the DCS console.
- Step 2** Click  in the upper left corner and select a region and a project.
- Step 3** In the navigation pane, choose **Cache Manager**.
- Step 4** Click the desired instance.
- Step 5** Choose **Performance Monitoring**. All monitoring metrics of the instance are displayed.

 NOTE

You can also click **View Metric** in the **Operation** column on the **Cache Manager** page. You will be redirected to the Cloud Eye console. The metrics displayed on the Cloud Eye console are the same as those displayed on the **Performance Monitoring** page of the DCS console.

----End

5.3 Configuring Alarm Rules for Critical Metrics

This section describes the alarm rules of some metrics and how to configure the rules. In actual scenarios, configure alarm rules for metrics by referring to the following alarm policies.

Alarm Policies for DCS Redis Instances

Table 5-6 DCS Redis instance metrics to configure alarm rules for

Metric	Normal Range	Alarm Policy	Approach Upper Limit	Handling Suggestion
CPU Usage	0-100	Alarm threshold: 70 Number of consecutive periods: 2 Alarm severity: Major	No	Consider capacity expansion based on the service analysis. The CPU capacity of a single-node or master/standby instance cannot be expanded. If you need larger capacity, use a cluster instance instead.
Memory Usage	0-100	Alarm threshold: 70 Number of consecutive periods: 2 Alarm severity: Major	No	Expand the capacity of the instance.


Metric	Normal Range	Alarm Policy	Approach Upper Limit	Handling Suggestion
Connected Clients	0-10,000	Alarm threshold: 8000 Number of consecutive periods: 2 Alarm severity: Major	No	Optimize the connection pool in the service code to prevent the number of connections from exceeding the maximum limit. For single-node and master/standby instances, the maximum number of connections allowed is 10,000. You can adjust the threshold based on service requirements.
New Connections (Count/min)	0-10,000	Alarm threshold: 10,000 Number of consecutive periods: 2 Alarm severity: Minor	-	Check whether connect is used and whether the client connection is abnormal. Use persistent connections ("pconnect" in Redis terminology) to ensure performance.
Input Flow	> 0	Alarm threshold: 80% of the assured bandwidth Number of consecutive periods: 2 Alarm severity: Major	Yes	Consider capacity expansion based on the service analysis and bandwidth limit. Configure this alarm only for single-node and master/standby DCS Redis 3.0 instances and set the alarm threshold to 80% of the assured bandwidth of DCS Redis 3.0 instances.

Metric	Normal Range	Alarm Policy	Approach Upper Limit	Handling Suggestion
Output Flow	> 0	Alarm threshold: 80% of the assured bandwidth Number of consecutive periods: 2 Alarm severity: Major	Yes	Consider capacity expansion based on the service analysis and bandwidth limit. Configure this alarm only for single-node and master/standby DCS Redis 3.0 instances and set the alarm threshold to 80% of the assured bandwidth of DCS Redis 3.0 instances.

Procedure


In the following example, an alarm rule is set for the **CPU Usage** metric.

Step 1 Log in to the DCS console.

Step 2 Click  in the upper left corner and select a region and a project.

Step 3 In the navigation pane, choose **Cache Manager**.

Step 4 In the same row as the DCS instance whose metrics you want to view, choose **More > View Metric**.

Step 5 Locate the **CPU Usage** metric. Hover over the metric and click  to create an alarm rule for the metric.

The **Create Alarm Rule** page is displayed.

Step 6 Specify the alarm rule details.

1. Specify the alarm policy and alarm severity.
2. Set the alarm notification configurations. If you enable **Alarm Notification**, set the validity period, notification object, and trigger condition.
3. Click **Next**.
4. Under **Specify Rules Name**, set the alarm name and description.
5. Click **Create**.

 **NOTE**

For more information about creating alarm rules, see the *Cloud Eye User Guide > Using the Alarm Function > Creating Alarm Rules*.

----End

6 Auditing

6.1 Operations That Can Be Recorded by CTS

With CTS, you can query, audit, and review operations performed on cloud resources. Traces include the operation requests sent using the management console or open APIs as well as the results of these requests.

The following lists the DCS operations that can be recorded by CTS.

Table 6-1 DCS operations that can be recorded by CTS

Operation	Resource Type	Trace Name
Creating an instance	DCS	createDCSInstance
Submitting an instance creation request	DCS	submitCreateDCSInstanceRequest
Deleting multiple instances	DCS	batchDeleteDCSInstance
Deleting an instance	DCS	deleteDCSInstance
Modifying instance information	DCS	modifyDCSInstanceInfo
Modifying instance configurations	DCS	modifyDCSInstanceConfig

Operation	Resource Type	Trace Name
Changing instance password	DCS	modifyDCSInstancePassword
Restarting an instance	DCS	restartDCSInstance
Submitting an instance restarting request	DCS	submitRestartDCSInstanceRequest
Starting an instance	DCS	startDCSInstance
Submitting an instance starting request	DCS	submitStartDCSInstanceRequest
Clearing instance data	DCS	flushDCSInstance
Restarting instances in batches	DCS	batchRestartDCSInstance
Submitting a request to restart instances in batches	DCS	submitBatchRestartDCSInstanceRequest
Starting multiple instances	DCS	batchStartDCSInstance
Submitting a request to start instances in batches	DCS	submitBatchStartDCSInstanceRequest
Restoring instance data	DCS	restoreDCSInstance
Submitting a request to restore instance data	DCS	submitRestoreDCSInstanceRequest


Operation	Resource Type	Trace Name
Backing up instance data	DCS	backupDCSInstance
Submitting a request to back up instance data	DCS	submitBackupDCSInstanceRequest
Deleting instance backup files	DCS	deleteInstanceBackupFile
Deleting background tasks	DCS	deleteDCSInstanceJobRecord
Modifying instance specifications	DCS	modifySpecification
Submitting a request to modify instance specifications	DCS	submitModifySpecificationRequest
Creating an instance subscription order	DCS	createInstanceOrder
Switching between master and standby nodes	DCS	masterStandbySwitchover
Resetting instance password	DCS	resetDCSInstancePassword
Submitting a request to clear instance data	DCS	submitFlushDCSInstanceRequest

6.2 Viewing Traces on the CTS Console

After CTS is enabled, the tracker starts recording operations on cloud resources. Operation records for the last seven days can be viewed on the CTS console. This section describes how to query operation records of the last seven days on the CTS console.

Procedure

Step 1 Log in to the management console.

Step 2 Click  in the upper left corner of the management console and select a region and a project.

 **NOTE**

Select the same region as your application service.

Step 3 Click **Service List** and choose **Management & Deployment > Cloud Trace Service**.

Step 4 In the navigation pane, click **Trace List**.

Step 5 Specify the filters used for querying traces. The following filters are available:

- **Search By:**

Select an option from the drop-down list. Select **DCS** from the **Trace Source** drop-down list.

When you select **Trace name**, you also need to select a specific trace name.

When you select **Resource ID**, you also need to select a specific resource ID.

When you select **Resource name**, you also need to select a specific resource name.

- **Operator:** Select a specific operator (a user other than tenant).
- **Trace Status:** Available options include **All trace status**, **normal**, **warning**, and **incident**. You can select only one of them.
- **Start time and end time:** You can specify the time period in which to query traces.


Step 6 Click  on the left of a trace to expand its details, as shown in [Figure 6-1](#).

Figure 6-1 Expanding trace details

Trace Name	Resource Type	Trace Source	Resource ID	Resource Name	Trace Status	Operator	Operation Time	Operation
createDCSInstanceS...	Redis	DCS	3980d1c8-c2f6-42cb-b8...		normal		04/16/2018 11:14:59 GMT+08:00	View Trace
Trace ID: 586d3349-4124-11e8-897d-2c790f6a4585		Trace Type: ConsoleAction		Source IP Address: [redacted]		Generated: 04/16/2018 11:14:59 GMT+08:00		

Step 7 Click **View Trace** in the **Operation** column. In the dialog box shown in [Figure 6-2](#), the trace structure details are displayed.

Figure 6-2 Viewing traces

```
{
  "time": "04/16/2018 11:14:59 GMT+08:00",
  "user": {
    "name": "██████████",
    "id": "5b64419de7f54010ace3ba9d63ece3c4",
    "domain": {
      "name": "██████████",
      "id": "cc9c0076188843ea938743dd166c7fef"
    }
  },
  "request": {
    "name": "██████████",
    "description": "",
    "engine": "Redis",
    "engine_version": "3.0.7",
    "capacity": 2,
    "password": "*****",
    "vpc_id": "47c7ec3a-181f-4c3d-930f-de255c330f8b",
    "security_group_id": "2907f342-5960-4513-b8a4-1ba31eceabc9",
    "subnet_id": "cb6cbaf3-ebf0-4e62-8793-570d2a6de24b",
    "available_zones": [
      "ae04cf9d61544df3806a3feeb401b204"
    ],
    "product_id": "00301-31100-0--0",
    "no_password_access": false,
    "maintain_begin": "02:00:00",
    "maintain_end": "02:00:00"
  }
}
```

----End

7 FAQs

7.1 Instance Types/Versions

7.1.1 Comparing Versions

When creating a DCS Redis instance, you can select the cache engine version and the instance type.

- **Version**

DCS supports Redis 3.0, 4.0, and 5.0. [Table 7-1](#) describes the differences between these versions. For more information about Redis 4.0 and 5.0 features, see sections "New Features of DCS for Redis 4.0 " and "New Features of DCS for Redis 5.0."

Table 7-1 Differences between Redis versions

Item	Redis 3.0	Redis 4.0 and Redis 5.0
Open-source compatibility	Redis 3.0.7	Redis 4.0.14 and 5.0.9, respectively
Instance deployment mode	Based on VMs	Containerized based on physical servers
CPU architecture	x86 and Arm	x86 and Arm
Time required for creating an instance	3–15 minutes, or 10–30 minutes for cluster instances.	8 seconds
QPS	100,000 QPS per node	100,000 QPS per node

Item	Redis 3.0	Redis 4.0 and Redis 5.0
Public network access	Supported	Not supported
Domain name connection	Supported within a VPC	Supported within a VPC
Visualized data management	Not supported	Web CLI for connecting to Redis and managing data
Instance types	Single-node, master/standby, and Proxy Cluster	Single-node, master/standby, and Redis Cluster
Instance total memory	Ranges from 2 GB to 1024 GB.	Regular specifications range from 2 GB to 1024 GB. Small specifications of 128 MB, 256 MB, 512 MB, and 1 GB are also available for single-node and master/standby instances.
Scale-up or scale-down	Online scale-up and scale-down	Online scale-up and scale-down
Backup and restoration	Supported for master/standby and cluster instances	Supported for master/standby and cluster instances

 **NOTE**

The underlying architectures vary by Redis version. Once a Redis version is chosen, it cannot be changed. For example, you cannot upgrade a DCS Redis 3.0 instance to Redis 4.0 or 5.0. If you require a higher Redis version, create a new instance that meets your requirements and then migrate data from the old instance to the new one.

- **Instance type**

DCS provides single-node, master/standby, Proxy Cluster, and Redis Cluster instance types. For details about their architectures and application scenarios, see section "DCS Instance Types".

7.1.2 New Features of DCS for Redis 4.0

Compared with DCS for Redis 3.0, DCS for Redis 4.0 and later versions add support for the new features of open-source Redis and supports faster instance creation.

Instance deployment changed from the VM mode to physical server-based containerization mode. An instance can be created within 8 to 10 seconds.

Redis 4.0 provides the following new features:

1. New commands, such as **MEMORY** and **SWAPDB**
2. Lazyfree, delaying the deletion of large keys and reducing the impact of the deletion on system resources
3. Memory performance optimization, that is, active defragmentation

MEMORY Command

In Redis 3.0 and earlier versions, you can execute the **INFO MEMORY** command to learn only the limited memory statistics. Redis 4.0 introduces the **MEMORY** command to help you better understand Redis memory usage.

```
127.0.0.1:6379[8]> memory help
1) MEMORY <subcommand> arg arg ... arg. Subcommands are:
2) DOCTOR - Return memory problems reports.
3) MALLOC-STATS -- Return internal statistics report from the memory allocator.
4) PURGE -- Attempt to purge dirty pages for reclamation by the allocator.
5) STATS -- Return information about the memory usage of the server.
6) USAGE <key> [SAMPLES <count>] -- Return memory in bytes used by <key> and its value. Nested values
are sampled up to <count>
> times (default: 5).
127.0.0.1:6379[8]>
```

usage

Enter **memory usage [key]**. If the key exists, the estimated memory used by the value of the key is returned. If the key does not exist, **nil** is returned.

```
127.0.0.1:6379[8]> set dcs "DCS is an online, distributed, in-memory cache service compatible with Redis,
and Memcached."
OK
127.0.0.1:6379[8]> memory usage dcs
(integer) 141
127.0.0.1:6379[8]>
```

NOTE

1. **usage** collects statistics on the memory usage of the value and the key, excluding the expire memory usage of the key.
// The following is verified based on Redis 5.0.2. Results may differ in other Redis versions.
192.168.0.66:6379> set a "Hello, world!"
OK
192.168.0.66:6379> memory usage a
(integer) 58
192.168.0.66:6379> set abc "Hello, world!"
OK
192.168.0.66:6379> memory usage abc
(integer) 60 //After the key name length changes, the memory usage also changes. This indicates that the usage statistics contain the usage of the key.
192.168.0.66:6379> expire abc 1000000
(integer) 1
192.168.0.66:6379> memory usage abc
(integer) 60 // After the expiration time is added, the memory usage remains unchanged. This indicates that the usage statistics do not contain the expire memory usage.
192.168.0.66:6379>
2. For hashes, lists, sets, and sorted sets, the **MEMORY USAGE** command samples statistics and provides the estimated memory usage.
Usage: **memory usage keyset samples 1000**
keyset indicates the key of a set, and *1000* indicates the number of samples.

stats

Returns the detailed memory usage of the current instance.

Usage: **memory stats**

```
127.0.0.1:6379[8]> memory stats
1) "peak.allocated"
2) (integer) 2412408
3) "total.allocated"
4) (integer) 2084720
5) "startup.allocated"
6) (integer) 824928
7) "replication.backlog"
... ..
```

The following table describes the meanings of some return items.

Table 7-2 memory stats

Return Value	Description
peak.allocated	Peak memory allocated by the allocator during Redis instance running. It is the same as used_memory_peak of info memory .
total.allocated	The number of bytes allocated by the allocator. It is the same as used_memory of info memory
startup.allocated	Initial amount of memory consumed by Redis at startup in bytes.
replication.backlog	Size in bytes of the replication backlog. It is specified in the repl-backlog-size parameter. The default value is 1 MB .
clients.slaves	The total size in bytes of all replicas overheads.
clients.normal	The total size in bytes of all clients overheads.
overhead.total	The sum of all overheads. overhead.total is the total memory total.allocated allocated by the allocator minus the actual memory used for storing data.
keys.count	The total number of keys stored across all databases in the server.
keys.bytes-per-key	Average number of bytes occupied by each key. Note that the overhead is also allocated to each key. Therefore, this value does not indicate the average key length.
dataset.bytes	Memory bytes occupied by Redis data, that is, overhead.total subtracted from total.allocated
dataset.percentage	The percentage of dataset.bytes out of the net memory usage.
peak.percentage	The percentage of peak.allocated out of total.allocated .
fragmentation	Memory fragmentation rate.

doctor

Usage: **memory doctor**

If the value of **used_memory (total.allocated)** is less than 5 MB, **MEMORY DOCTOR** considers that the memory usage is too small and does not perform further diagnosis. If any of the following conditions is met, Redis provides diagnosis results and suggestions:

1. The peak allocated memory is greater than 1.5 times of the current **total_allocated**, that is, **peak.allocated/total.allocated** > 1.5, indicating that the memory fragmentation rate is high, and that the RSS is much larger than **used_memory**.
2. The value of high fragmentation/fragmentation is greater than 1.4, indicating that the memory fragmentation rate is high.
3. The average memory usage of each normal client is greater than 200 KB, indicating that the pipeline may be improperly used or the Pub/Sub client does not process messages in time.
4. The average memory usage of each slave client is greater than 10 MB, indicating that the write traffic of the master is too high.

purge

Usage: **memory purge**

Executes the **jemalloc** internal command to release the memory. The released objects include the memory that is occupied but not used by Redis processes, that is, memory fragments.

NOTE

MEMORY PURGE applies only to the Redis instance that uses **jemalloc** as the allocator.

Lazyfree

Problem

Redis is single-thread. When a time-consuming request is executed, all requests are queued. Before the request is completed, Redis cannot respond to other requests. As a result, performance problems may occur. One of the time-consuming requests is deleting a large key.

Principle

The Lazyfree feature of Redis 4.0 avoids the blockage caused by deleting large keys, ensuring performance and availability.

When deleting a key, Redis asynchronously releases the memory occupied by the key. The key release operation is processed in the sub-thread of the background I/O (BIO).

Usage

1. Active deletion
 - **unlink**
Similar to **DEL**, this command removes keys. If there are more than 64 elements to be deleted, the memory release operation is executed in an

independent BIO thread. Therefore, the **UNLINK** command can delete a large key containing millions of elements in a short time.

- **flushall/flushdb**

An **ASYNC** option was added to **FLUSHALL** and **FLUSHDB** in order to let the entire dataset or a single database to be freed asynchronously.

2. Passive deletion: deletion of expired keys and eviction of large keys

There are four scenarios for passive deletion and each scenario corresponds to a parameter. These parameters are disabled by default.

```
lazyfree-lazy-eviction no // Whether to enable Lazyfree when the Redis memory usage reaches
maxmemory and the eviction policy is set.
lazyfree-lazy-expire no // Whether to enable Lazyfree when the key with TTL is going to expire.
lazyfree-lazy-server-del no // An implicit DEL key is used when an existing key is processed.
slave-lazy-flush no // Perform full data synchronization for the standby node. Before loading the RDB
file of the master, the standby node executes the FLUSHALL command to clear its own data.
```

 **NOTE**

To enable these configurations, contact technical support.

Other New Commands

1. **swapdb**

Swaps two Redis databases.

```
swapdb dbindex1 dbindex2
```

2. **zlexcount**

Returns the number of elements in the sorted set.

```
zlexcount key min max
```

Memory and Performance Optimization

1. Compared to before, the same amount of data can be stored with less memory.
2. Used memory can be defragmented and gradually evicted.

7.1.3 New Features of DCS for Redis 5.0

DCS for Redis 5.0 is compatible with the new features of the open-source Redis 5.0, in addition to all the improvements and new commands in Redis 4.0.

Stream Data Structure

Stream is a new data type introduced with Redis 5.0. It supports message persistence and multicast.

Figure 7-1 shows the structure of a Redis stream, which allows messages to be appended to the stream.

Streams have the following features:

1. A stream can have multiple consumer groups.
2. Each consumer group contains a **Last_delivered_id** that points to the last consumed item (message) in the consumer group.

3. Each consumer group contains multiple consumers. All consumers share the **last_delivered_id** of the consumer group. A message can be consumed by only one consumer.
4. **pending_ids** in the consumer can be used to record the IDs of items that have been sent to the client, but have not been acknowledged.
5. For detailed comparison between stream and other Redis data structures, see [Table 7-3](#).

Figure 7-1 Stream data structure

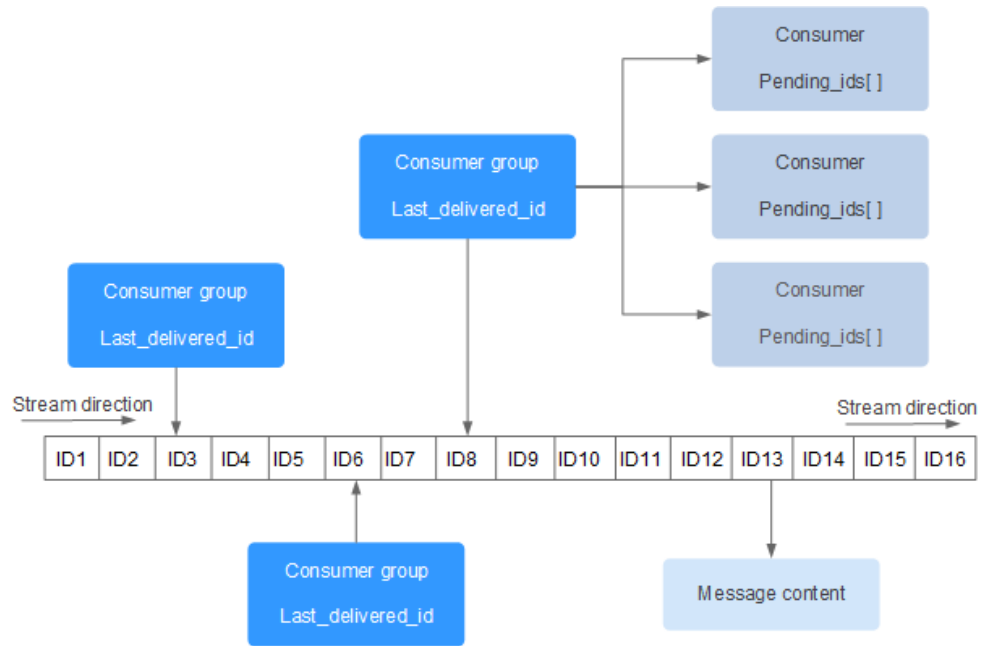


Table 7-3 Differences between streams and existing Redis data structures

Item	Stream	List, Pub/Sub, Zset
Complexity of seeking items	$O(\log(N))$	List: $O(N)$
Offset	Supported. Each item has a unique ID. The ID is not changed as other items are added or evicted.	List: Not supported. If an item is evicted, the latest item cannot be located.
Persistence	Supported. Streams are persisted to AOF and RDB files.	Pub/Sub: Not supported.
Consumer group	Supported.	Pub/Sub: Not supported.
Acknowledgment	Supported.	Pub/Sub: Not supported.

Item	Stream	List, Pub/Sub, Zset
Performance	Not related to the number of consumers.	Pub/Sub: Positively related to the number of clients.
Eviction	Streams are memory efficient by blocking to evict the data that is too old and using a radix tree and listpack.	Zset consumes more memory because it does not support inserting same items, blocking, or evicting data
Randomly deleting items	Not supported.	Zset: Supported.

Stream commands

Stream commands are described below in the order they are used. For details, see [Table 7-4](#).

1. Run the **XADD** command to add a stream item, that is, create a stream. The maximum number of messages that can be saved can be specified when adding the item.
2. Create a consumer group by running the **XGROUP** command.
3. A consumer uses the **XREADGROUP** command to consume messages.
4. After the consumption, the client runs the **XACK** command to confirm that the consumption is successful.

Figure 7-2 Stream commands

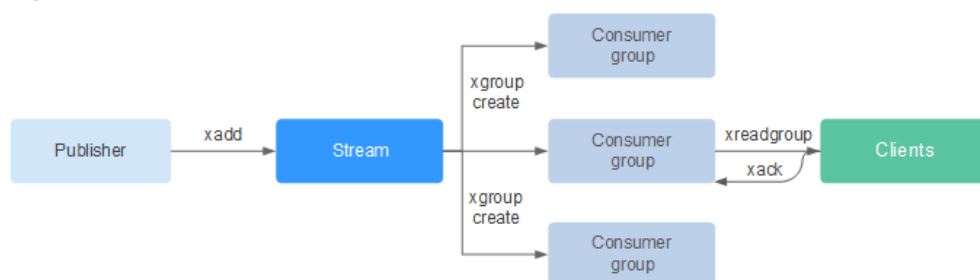


Table 7-4 Stream commands description

Command	Description	Syntax
XACK	Deletes one or multiple messages from the <i>pending entry list</i> (PEL) a consumer group of the stream.	XACK key group ID [ID ...]

Command	Description	Syntax
XADD	Adds a specified entry to the stream at a specified key. If the key does not exist, running this command will result in a key to be automatically created based on the entry.	XADD key ID field string [field string ...]
XCLAIM	Changes the ownership of a pending message, so that the new owner is the consumer specified as the command argument.	XCLAIM key group consumer min-idle-time ID [ID ...] [IDLE ms] [TIME ms-unix-time] [RETRYCOUNT count] [FORCE] [JUSTID]
XDEL	Removes the specified entries from a stream, and returns the number of entries deleted, that may be different from the number of IDs passed to the command in case certain IDs do not exist.	XDEL key ID [ID ...]
XGROUP	Manages the consumer groups associated with a stream. You can use XGROUP to: <ul style="list-style-type: none"> • Create a new consumer group associated with a stream. • Destroy a consumer group. • Remove a specified consumer from a consumer group. • Set the consumer group <i>last delivery ID</i> to something else. 	XGROUP [CREATE key groupname id-or-\$] [SETID key id-or-\$] [DESTROY key groupname] [DELCONSUMER key groupname consumername]
XINFO	Retrieves different information about the streams and associated consumer groups.	XINFO [CONSUMERS key groupname] key key [HELP]
XLEN	Returns the number of entries in a stream. If the specified key does not exist, 0 is returned, indicating an empty stream.	XLEN key
XPENDING	Obtains data from a stream through a consumer group. This command is the interface to inspect the list of pending messages in order to observe and understand what clients are active, what messages are pending to be consumed, or to see if there are idle messages.	XPENDING key group [start end count] [consumer]

Command	Description	Syntax
XRANGE	Returns entries matching a given range of IDs.	XRANGE key start end [COUNT count]
XREAD	Reads data from one or multiple streams, only returning entries with an ID greater than the last received ID reported by the caller.	XREAD [COUNT count] [BLOCK milliseconds] STREAMS key [key ...] ID [ID ...]
XREADGROUP	A special version of the XREAD command, which is used to specify a consumer group to read from.	XREADGROUP GROUP group consumer [COUNT count] [BLOCK milliseconds] STREAMS key [key ...] ID [ID ...]
XREVRANGE	This command is exactly like XRANGE , but with the notable difference of returning the entries in reverse order, and also taking the start-end range in reverse order.	XREVRANGE key end start [COUNT count]
XTRIM	Trims the stream to a specified number of items, if necessary, evicting old items (items with lower IDs).	XTRIM key MAXLEN [~] count

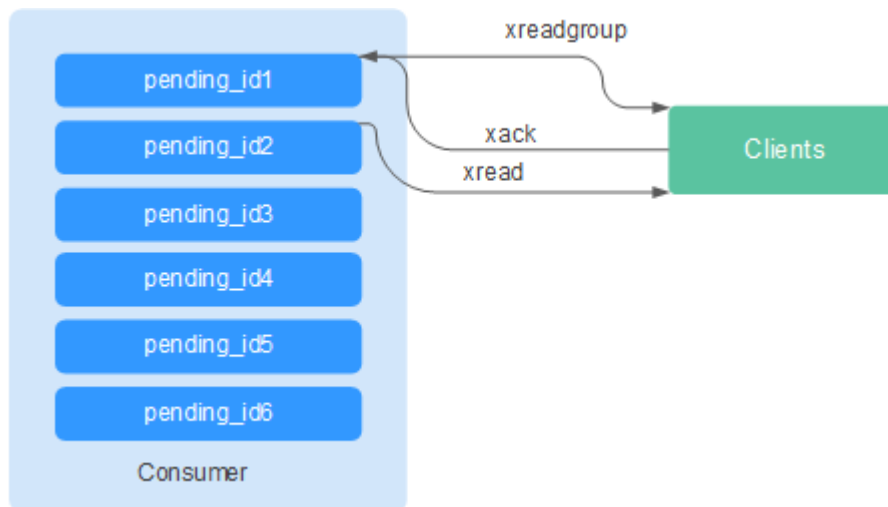
Message (stream item) acknowledgement

Compared with Pub/Sub, streams not only support consumer groups, but also message acknowledgement.

When a consumer invokes the **XREADGROUP** command to read or invokes the **XCLAIM** command to take over a message, the server does not know whether the message is processed at least once. Therefore, once having successfully processed a message, the consumer should invoke the **XACK** command to notify the stream so that the message will not be processed again. In addition, the message is removed from PEL and the memory will be released from the Redis server.

In some cases, such as network faults, the client does not invoke **XACK** after consumption. In such cases, the item ID is retained in PEL. After the client is reconnected, set the start message ID of **XREADGROUP** to 0-0, indicating that all PEL messages and messages after **last_id** are read. In addition, repeated message transmission must be supported when consumers consume messages.

Figure 7-3 Acknowledgment mechanism



Memory Usage Optimization

The memory usage of Redis 5.0 is optimized based on the previous version.

- Active defragmentation

If a key is modified frequently and the value length changes constantly, Redis will allocate additional memory for the key. To achieve high performance, Redis uses the memory allocator to manage memory. Memory is not always freed up to the OS. As a result, memory fragments occur. If the fragmentation ratio (**used_memory_rss/used_memory**) is greater than 1.5, the memory usage is inefficient.

To reduce memory fragments, properly plan and use cache data and standardize data writing.

For Redis 3.0 and earlier versions, memory fragmentation problems are resolved by restarting the process regularly. It is recommended that the actual cache data does not exceed 50% of the available memory.

For Redis 4.0, active defragmentation is supported, and memory is defragmented while online. In addition, Redis 4.0 supports manual memory defragmentation by running the **memory purge** command.

For Redis 5.0, improved active defragmentation is supported with the updated Jemalloc, which is faster, more intelligent, and provides lower latency.

- HyperLogLog implementation improvements

A HyperLogLog is a probabilistic data structure used to calculate the cardinality of a set while consuming little memory. Redis 5.0 improves HyperLogLog by further optimizing its memory usage.

For example: the B-tree is efficient in counting, but consumes a lot of memory. By using HyperLogLog, a lot of memory can be saved. While the B-tree requires 1 MB memory for counting, HyperLogLog needs only 1 KB.

- Enhanced memory statistics

The information returned by the **INFO** command is more detailed.

New and Better Commands

1. Enhanced client management

- redis-cli supports cluster management.

In Redis 4.0 and earlier versions, the **redis-trib** module needs to be installed to manage clusters.

Redis 5.0 optimizes redis-cli, integrating all cluster management functions. You can run the **redis-cli --cluster help** command for more information.

- The client performance is enhanced in frequent connection and disconnection scenarios.

This optimization is valuable when your application needs to use short connections.

2. Simpler use of sorted sets

ZPOPMIN and **ZPOPMAX** commands are added for sorted sets.

- ZPOPMIN key [count]

Removes and returns up to **count** members with the lowest scores in the sorted set stored at **key**. When returning multiple elements, the one with the lowest score will be the first, followed by the elements with higher scores.

- ZPOPMAX key [count]

Removes and returns up to **count** members with the highest scores in the sorted set stored at **key**. When returning multiple elements, the one with the lowest score will be the first, followed by the elements with lower scores.

3. More sub-commands added to the help command

The **help** command can be used to view help information, saving you the trouble of visiting **redis.io** every time. For example, run the following command to view the stream help information: **xinfo help**

```
127.0.0.1:6379> xinfo help
1) XINFO <subcommand> arg arg ... arg. Subcommands are:
2) CONSUMERS <key> <groupname> -- Show consumer groups of group <groupname>.
3) GROUPS <key> -- Show the stream consumer groups.
4) STREAM <key> -- Show information about the stream.
5) HELP -- Print this help.
127.0.0.1:6379>
```

4. redis-cli command input tips

After you enter a complete command, redis-cli displays a parameter tip to help you memorize the syntax format of the command.

As shown in the following figure, run the **zadd** command, and redis-cli displays **zadd** syntax in light color.

```
# Cluster
cluster_enabled:0

# Keyspace
db0:keys=1,expires=0,avg_ttl=0
198.19.59.199:6379> zadd key [NX|XX] [CH] [INCR] score member [score member ...]
```

RDB Storing LFU and LRU Information

In Redis 5.0, storage key eviction policies **LRU** and **LFU** were added to the RDB snapshot file.

- FIFO: First in, first out. The earliest stored data is evicted first.
- LRU: Least recently used. Data that is not used for a long time is evicted first.
- LFU: Least frequently used. Data that is least frequently used is evicted first.

NOTE

The RDB file format of Redis 5.0 is modified and is backward compatible. Therefore, if a snapshot is used for migration, data can be migrated from the earlier Redis versions to Redis 5.0, but cannot be migrated from the Redis 5.0 to the earlier versions.

7.2 Client and Network Connection

7.2.1 Security Group Configurations

This section describes how to configure a security group for accessing a DCS instance **within a VPC**.

An ECS can communicate with a DCS instance if they belong to the same VPC and security group rules are configured correctly.

In addition, you must configure correct rules for the security groups of both the ECS and DCS instance so that you can access the instance through your client.

- If the ECS and DCS instance are configured with the same security group, network access in the group is not restricted by default.
- If the ECS and DCS instance are configured with different security groups, add security group rules to ensure that the ECS and DCS instance can access each other.

NOTE

- Suppose that the ECS on which the client runs belongs to security group **sg-ECS**, and the DCS instance that the client will access belongs to security group **sg-DCS**.
 - Suppose that the port number of the DCS service is 6379.
 - The remote end is a security group or an IP address.
- a. Configuring security group for the ECS.
Add the following outbound rule to allow the ECS to access the DCS instance. Skip this rule if there are no restrictions on the outbound traffic.
 - b. Configuring security group for the DCS instance.
To ensure that your client can access the DCS instance, add the following inbound rule to the security group configured for the DCS instance:

NOTICE

For the source IP address, use the specified IP address of the DCS instance. Avoid using **0.0.0.0/0** to prevent ECSs bound with the same security group from being attacked by Redis vulnerability exploits.

7.2.2 Does DCS Support Public Access?

No. DCS instances cannot be accessed at their EIPs over public networks. To ensure security, the ECS that serves as a client and the DCS instance that the client will access must belong to the same VPC.

In the application development and debugging phase, you can also use an SSH agent to access DCS instances in the local environment.

7.2.3 Does DCS Support Cross-VPC Access?

Cross-VPC means the client and the instance are not in the same VPC.

Generally, VPCs are isolated from each other and ECSs cannot access DCS instances that belong to a different VPC from these ECSs.

However, by establishing VPC peering connections between VPCs, ECSs can access single-node and master/standby DCS instances across VPCs.

When using VPC peering connections to access DCS instances across VPCs, adhere to the following rules:

- If network segments 172.16.0.0/12 to 172.16.0.0/24 are used during DCS instance creation, the client cannot be in any of the following network segments: 192.168.1.0/24, 192.168.2.0/24, and 192.168.3.0/24.
- If network segments 192.168.0.0/16 to 192.168.0.0/24 are used during DCS instance creation, the client cannot be in any of the following network segments: 172.31.1.0/24, 172.31.2.0/24, and 172.31.3.0/24.
- If network segments 10.0.0.0/8 to 10.0.0.0/24 are used during DCS instance creation, the client cannot be in any of the following network segments: 172.31.1.0/24, 172.31.2.0/24, and 172.31.3.0/24.

For more information about VPC peering connection, see "VPC Peering Connection" in *Virtual Private Cloud User Guide*.

NOTICE

Cluster DCS Redis instances do not support cross-VPC access. ECSs in a VPC cannot access cluster DCS instances in another VPC by using VPC peering connections.

7.2.4 What Should I Do If Access to DCS Fails After Server Disconnects?

Analysis: If persistent connections ("pconnect" in Redis terminology) or connection pooling is used and connections are closed after being used for connecting to DCS instances, errors will be returned at attempts to reuse the connections.

Solution: When using pconnect or connection pooling, do not close the connection after the end of a request. If the connection is dropped, re-establish it.

7.2.5 Why Do Requests Sometimes Time Out in Clients?

Occasional timeout errors are normal because of network connectivity and client timeout configurations.

You are advised to include reconnection operations into your service code to avoid service failure if a single request fails.

If timeout errors occur frequently, contact O&M personnel.

7.2.6 What Should I Do If an Error Is Returned When I Use the Jedis Connection Pool?

The error message that will possibly be displayed when you use the Jedis connection pool is as follows:

```
redis.clients.jedis.exceptions.JedisConnectionException: Could not get a resource from the pool
```

If this error message is displayed, check whether your instance is running properly. If it is running properly, perform the following checks:

Step 1 Network

1. Check the IP address configurations.

Check whether the IP address configured on the Jedis client is the same as the subnet address configured for your DCS instance.

2. Test the network.

Use the ping command and telnet on the client to test the network.

- If the network cannot be pinged:

For intra-VPC access, ensure that the client and your DCS instance belong to the same VPC and security group, or the security group of your DCS instance allows access through port 6379. For details, see [Security Group Configurations](#).

- If the IP address can be pinged but telnet failed, restart your instance. If the problem persists after the restart, contact technical support.

Step 2 Check the number of connections.

Check whether the number of established network connections exceeds the upper limit configured for the Jedis connection pool. If the number of established connections approaches the configured upper limit, restart the DCS service and check whether the problem persists. If the number of established connections is far below the upper limit, continue with the following checks.

In Unix or Linux, run the following command to query the number of established network connections:

```
netstat -an | grep 6379 | grep ESTABLISHED | wc -l
```

In Windows, run the following command to query the number of established network connections:

```
netstat -an | find "6379" | find "ESTABLISHED" /C
```

Step 3 Check the JedisPool code.

If the number of established connections approaches the upper limit, determine whether the problem is caused by service concurrency or incorrect usage of JedisPool.

When using JedisPool, you must call `jedisPool.returnResource()` or `jedis.close()` (recommended) to release the resources after you call `jedisPool.getResource()`.

Step 4 Check the number of TIME_WAIT connections.

Run the `ss -s` command to check whether there are too many TIME_WAIT connections on the client.

```
root@heru-nodelite:~# ss -s
Total: 140 (kernel 240)
TCP: 11 (estab 3, closed 1, orphaned 0, synrecv 0, timewait 0/0), ports 0

Transport Total      IP        IPv6
*          240      -        -
RAW        0         0         0
UDP        2         2         0
TCP        10        6         4
INET       12        8         4
FRAG       0         0         0
```

If there are too many TIME_WAIT connections, modify the kernel parameters by running the `/etc/sysctl.conf` command as follows:

```
##Uses cookies to prevent some SYN flood attacks when the SYN waiting queue overflows.
net.ipv4.tcp_syncookies = 1
##Reuses TIME_WAIT sockets for new TCP connections.
net.ipv4.tcp_tw_reuse = 1
##Enables quick reclamation of TIME_WAIT sockets in TCP connections.
net.ipv4.tcp_tw_recycle = 1
##Modifies the default timeout time of the system.
net.ipv4.tcp_fin_timeout = 30
```

After the modification, run the `/sbin/sysctl -p` command for the modification to take effect.

Step 5 If the problem persists after you perform the preceding checks, perform the following steps.

Capture packets and send packet files along with the time and description of the exception to technical support for analysis.

Run the following command to capture packets:

```
tcpdump -i eth0 tcp and port 6379 -n -nn -s 74 -w dump.pcap
```

In Windows, you can also install the Wireshark tool to capture packets.

NOTE

Replace the NIC name to the actual one.

----End

7.2.7 Why Is "ERR unknown command" Displayed When I Access a DCS Redis Instance Through a Redis Client?

The possible causes are as follows:

1. The command is spelled incorrectly.

As shown in the following figure, the error message is returned because the correct command for deleting a string should be **del**.

```
192.168.0.244:6379> delete hellokitty
(error) ERR unknown command 'delete'
192.168.0.244:6379> del hellokitty
(integer) 1
192.168.0.244:6379>
```

2. A command available in a higher Redis version is run in a lower Redis version.

As shown in the following figure, the error message is returned because a stream command (available in Redis 5.0) is run in Redis 3.0.

```
192.168.0.244:6379> xadd stream01 * field01 teststring
(error) ERR unknown command 'xadd'
192.168.0.244:6379> info server
# Server
redis_version:3.0.7.9
redis_git_sha1:10fba618
```

3. Some commands are disabled.

DCS Redis instance interfaces are fully compatible with the open-source Redis in terms of data access. However, for ease of use and security purposes, some operations cannot be initiated through Redis clients. For details about disabled commands, see [Command Compatibility](#).

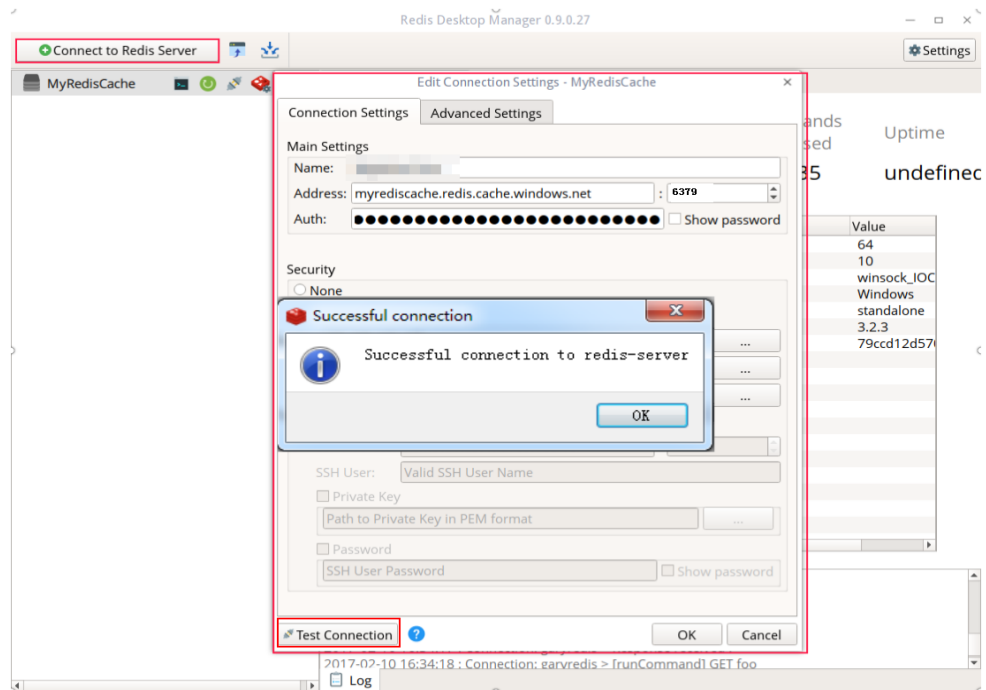
7.2.8 How Do I Access a DCS Redis Instance Through Redis Desktop Manager?

You can access a DCS Redis instance through the Redis Desktop Manager within a VPC.

1. Enter the address, port number (6379), and authentication password of the DCS instance you want to access.
2. Click **Test Connection**.

The system displays a success message if the connection is successful.

Figure 7-4 Accessing a DCS Redis instance through Redis Desktop Manager over the intranet



NOTE

When accessing a cluster DCS instance, the Redis command is run properly, but an error message may display on the left because DCS clusters are based on Codis, which differs from the native Redis in terms of the **INFO** command output.

7.2.9 What If "ERR Unsupported CONFIG subcommand" is Displayed in SpringCloud?

By using DCS Redis instances, Spring Session can implement session sharing. When interconnecting with Spring Cloud, the following error information is displayed:

Figure 7-5 Spring Cloud error information

```
org.springframework.dao.InvalidDataAccessApiUsageException: ERR Unsupported CONFIG subcommand; nested exception is redis.clients.jedis.exceptions.JedisDataException: ERR Unsupported CONFIG subcommand
2019-02-01 00:36:59 INFO com.alibaba.druid.pool.DruidDataSource - {dataSource-2} closed
2019-02-01 00:36:59 INFO com.alibaba.druid.pool.DruidDataSource - {dataSource-1} closed
2019-02-01 00:36:59 ERROR org.springframework.web.context.ContextLoader - Context initialization failed
org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'enableRedisKeyspaceNotificationsInitializer' defined in class path resource [org.springframework/session/data/annotation/web/http/RedisHttpSessionConfiguration.class]: Invocation of init method failed; nested exception is org.springframework.dao.InvalidDataAccessApiUsageException: ERR Unsupported CONFIG subcommand; nested exception is redis.clients.jedis.exceptions.JedisDataException: ERR Unsupported CONFIG subcommand
    at org.springframework.beans.factory.support.AbstractAutowiredBeanFactory.initializeBean(AbstractAutowiredBeanFactory.java:1794)
    at org.springframework.beans.factory.support.AbstractAutowiredBeanFactory.doCreateBean(AbstractAutowiredBeanFactory.java:583)
    at org.springframework.beans.factory.support.AbstractBeanFactory.lambda$doGetBean$0(AbstractBeanFactory.java:312)
    at org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleton(DefaultSingletonBeanRegistry.java:228)
    at org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(AbstractBeanFactory.java:318)
    at org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBeanFactory.java:288)
    at org.springframework.context.support.AbstractApplicationContext.finishBeanFactoryInitialization(AbstractApplicationContext.java:756)
    at org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:450)
```

For security purposes, DCS does not support the **CONFIG** command initiated by a client. You need to perform the following steps:

1. On the DCS console, set the value of the **notify-keyspace-event** parameter to **Egx** for a DCS Redis instance.
2. Add the following content to the XML configuration file of the Spring framework:

```
<util:constant  
static-  
field="org.springframework.session.data.redis.config.ConfigureRedisAction.NO  
_OP"/>
```

3. Modify the related Spring code. Enable the **ConfigureRedisAction.NO_OP** bean component to forbid a client to invoke the **CONFIG** command.

```
@Bean  
public static ConfigureRedisAction configureRedisAction() {  
    return ConfigureRedisAction.NO_OP;  
}
```

For more information, see the [Spring Session Documentation](#).

NOTICE

Session sharing is supported only by **single-node** and **master/standby** DCS Redis instances, but not by cluster DCS Redis instances.

7.2.10 How Do I Troubleshoot Redis Connection Failures?

Preliminary checks:

- Check the connection address.
Obtain the connection address from the instance basic information page on the DCS console.
- Check the instance password.
If the instance password is incorrect, the port can still be accessed but the authentication will fail.
- Check the port.
Port 6379 is the default port used in intra-VPC access to a DCS Redis instance.
- Check if the maximum bandwidth has been reached.
If the bandwidth reaches the maximum bandwidth for the corresponding instance specifications, Redis connections may time out.
- Check the inbound access rules of the security group.
Intra-VPC access: If the Redis client and the Redis instance are bound with different security groups, allow inbound access over port 6379 for the security group of the instance.
For details, see [Security Group Configurations](#).
- Check the configuration parameter **notify-keyspace-events**.
Set **notify-keyspace-events** to **Egx**.

Further checks:

- Jedis connection pool error
- Error "Read timed out" or "Could not get a resource from the pool"

Check if the **KEYS** command has been used. This command consumes a lot of resources and can easily block Redis. Instead, use the **SCAN** command and avoid executing the command frequently.

7.2.11 What Should Be Noted When Using Redis for Pub/Sub?

Pay attention to the following issues when using Redis for pub/sub:

- Your client must process messages in a timely manner.
Your client subscribes to a channel. If it does not receive messages in a timely manner, DCS instance messages may be overstocked. If the size of accumulated messages reaches the threshold (32 MB by default) or remains at a certain level (8 MB by default) for a certain period of time (1 minute by default), your client will be automatically disconnected to prevent server memory exhaustion.
- Your client must support connection re-establishment in case of disconnection.
In the event of a disconnection, you need to run the **subscribe** or **psubscribe** command on your client to subscribe to a channel again. Otherwise, your client cannot receive messages.
- Do not use pub/sub in scenarios with high message reliability requirements.
The Redis pub/sub is not a reliable messaging system. Messages that are not retrieved will be discarded when your client is disconnected or a master/standby switchover occurs.

7.3 Redis Usage

7.3.1 Why Is CPU Usage of a DCS Redis Instance 100%?

- Possible cause 1:
The service QPS is so high that the CPU usage spikes to 100%.
- Possible cause 2:
You have run commands that consume a lot of resources, such as **KEYS**. This will make CPU usage spike and can easily trigger a master/standby switchover.

7.3.2 Can I Change the VPC and Subnet for a DCS Redis Instance?

No. Once an instance is created, its VPC and subnet cannot be changed. If you want to use a different set of VPC and subnet, create a same instance and specify a desired set of VPC and subnet. After the new instance is created, you can migrate data from the old instance to the new instance by following the [data migration instructions](#).

7.3.3 Why Aren't Security Groups Configured for DCS Redis 4.0 and 5.0 Instances?

Currently, DCS Redis 4.0 and Redis 5.0 instances use VPC endpoints and do not support security groups.

7.3.4 Do DCS Redis Instances Limit the Size of a Key or Value?

- The maximum allowed size of a key is 512 MB.
To reduce memory usage and facilitate key query, ensure that each key does not exceed 1 KB.
- The maximum allowed size of a string is 512 MB.
- The maximum allowed size of a Set, List, or Hash is 512 MB.
In essence, a Set is a collection of Strings; a List is a list of Strings; a Hash contains mappings between string fields and string values.

Prevent the client from constantly writing large values in Redis. Otherwise, network transmission efficiency will be lowered and the Redis server would take a longer time to process commands, resulting in higher latency.

7.3.5 Can I Obtain the Addresses of the Nodes in a Cluster DCS Redis Instance?

Cluster DCS Redis 3.0 instances (Proxy Cluster type) are used in the same way that you use single-node or master/standby instances. You do not need to know the backend node addresses.

For a cluster DCS Redis 4.0 or 5.0 instance (Redis Cluster type), run the **CLUSTER NODES** command to obtain node addresses:

```
redis-cli -h {redis_address} -p {redis_port} -a {redis_password} cluster nodes
```

In the output similar to the following, obtain the IP addresses and port numbers of all the master nodes.

```
[root@ecs-54-centos ~]# redis-cli -h 192.168.0.140 -p 6379 -a 23 cluster nodes
fb75f0743af4695a3d241ff7790b2f508e4985ff 192.168.0.140:6379@16379 myself,master - 0 1562144170000 3 connected
d112bae791b2bbd9602fe32963536b8a0db9eb79 192.168.0.61:6379@16379 master - 0 1562144171524 1 connected 0-5460
73e2f8fe196166f9ad1283361867d24c136413f0 192.168.0.194:6379@16379 master - 0 1562144170000 2 connected 5461-1040d72299fde6045de0f79ee4b97910b505acbc6a 192.168.0.231:6379@16379 slave 73e2f8fe196166f9ad1283361867d24c136413
be6c07Faa64d724323e0d7cedc3f38346dcbd212 192.168.0.80:6379@16379 slave fb75f0743af4695a3d241ff7790b2f508e4985f
c16b9acaeeed7dd0721f129596cd43bd499c0e396 192.168.0.169:6379@16379 slave d112bae791b2bbd9602fe32963536b8a0db9eb
```

7.3.6 Why Is Available Memory of a DCS Redis 3.0 Instance Smaller Than Instance Cache Size?

DCS Redis 3.0 instances are deployed on VMs and some memory is reserved for system overheads.

7.3.7 Does DCS for Redis Support Multiple Databases?

Both single-node and master/standby DCS Redis instances support multiple databases. By default, single-node and master/standby DCS instances can read and write data in 256 databases (databases numbering 0–255).

Cluster DCS instances do not support data read/write in multiple databases.

7.3.8 Does DCS for Redis Support Redis Clusters?

Yes. DCS for Redis 4.0 and 5.0 support Redis Clusters. DCS for Redis 3.0 supports Proxy Clusters.

7.3.9 Does DCS for Redis Support Sentinel?

Yes. Redis Sentinel is supported by DCS for Redis 4.0 and 5.0 and is enabled by default. Sentinel constantly checks if master and replica nodes are running properly. If the master is not running properly, Sentinel starts a failover process and promotes a replica to master.

However, DCS for Redis 3.0 does not support Redis Sentinel. Instead, it uses keepalive to monitor master and replica nodes and to manage failovers.

7.3.10 What Is the Default Data Eviction Policy?

Data is evicted from cache based on a user-defined space limit in order to make space for new data. In the current versions of DCS, you can select an eviction policy.

noeviction is the default eviction policy for single-node and master/standby DCS Redis instances. You can change the eviction policy by configuring the instance parameters on the DCS console.

volatile-lru is the default eviction policy for cluster DCS Redis instances. To change the eviction policy for cluster instances, contact technical support.

When **maxmemory** is reached, you can select one of the following eight eviction policies:

- **noeviction**: When the memory limit is reached, DCS instances return errors to clients and no longer process write requests and other requests that could result in more memory to be used. However, **DEL** and a few more exception requests can continue to be processed.
- **allkeys-lru**: DCS instances try to evict the least recently used keys first, in order to make space for new data.
- **volatile-lru**: DCS instances try to evict the least recently used keys with an expire set first, in order to make space for new data.
- **allkeys-random**: DCS instances recycle random keys so that new data can be stored.
- **volatile-random**: DCS instances evict random keys with an expire set, in order to make space for new data.
- **volatile-ttl**: DCS instances evict keys with an expire set, and try to evict keys with a shorter time to live (TTL) first, in order to make space for new data.
- **allkeys-lfu**: DCS instances evict the least frequently used keys from all keys.
- **volatile-lfu**: DCS instances evict the least frequently used keys with an **expire** field from all keys.

NOTE

If no key can be recycled, **volatile-lru**, **volatile-random**, and **volatile-ttl** are the same as **noeviction**. For details, see the description of **noeviction**.

7.3.11 What Should I Do If an Error Occurs in Redis Exporter?

Start the Redis exporter using the CLI. Based on the output, check for errors and troubleshoot accordingly.

```
root@ecs-gangchao-ubuntu:~/inc# ./redis_exporter -redis.addr 172.18.0.32:6379
INFO[0000] Redis Metrics Exporter v0.25.0 build date: 2018-12-22-18:05:37 sha1: 1b148498f01340e58335299eca42e2a8095397e5
Go: go1.11.4
INFO[0000] Providing metrics at :9121/metrics
INFO[0000] Connecting to redis hosts: [string("172.18.0.32:6379")]
INFO[0000] Using alias: [string("")]
```

7.3.12 Why Is Memory Usage More Than 100%?

This is normal due to Redis functions (such as master/replica replication and lazyfree). When the memory becomes full, scale up the instance or remove unnecessary data.

7.3.13 Why Is Redisson Distributed Lock Not Supported by DCS Proxy Cluster Redis 3.0 Instances?

Redisson implements lock acquisition and unlocking in the following process:

1. Redisson lock acquisition and unlocking are implemented by running Lua scripts.
2. During lock acquisition, the **EXISTS**, **HSET**, **PEXPIRE**, **HEXISTS**, **HINCRBY**, **PEXPIRE**, and **PTTL** commands must be executed in the Lua script.
3. During unlocking, the **EXISTS**, **PUBLISH**, **HEXISTS**, **PEXPIRE**, and **DEL** commands must be executed in the Lua script.

In a proxy-based cluster, the proxy processes **PUBLISH** and **SUBSCRIBE** commands and forwards requests to the Redis server. The **PUBLISH** command cannot be executed in the Lua script.

As a result, Proxy Cluster DCS Redis 3.0 instances do not support Redisson distributed locks. To use Redisson, resort to Redis 4.0 or 5.0 instead.

7.3.14 Can I Customize or Change the Port for Accessing a DCS Instance?

You cannot customize or change the port for accessing a DCS Redis 3.0 or Memcached instance. You can customize and change the port for accessing a DCS Redis 4.0 or 5.0 instance.

- Redis 3.0
Use port 6379 for intra-VPC access.
- Memcached
Use port 11211 for intra-VPC access. Public access is not supported.
- Redis 4.0 and Redis 5.0
You can specify a port (ranging from 1 to 65535) or use the default port (6379) for accessing a DCS Redis 4.0 or 5.0 instance. If no port is specified, the default port will be used.

If the instance and the client use different security groups, you must configure access rules for the security groups, allowing access through the specified port. For details, see [Security Group Configurations](#).

7.3.15 Can I Modify the Connection Addresses for Accessing a DCS Instance?

After a DCS instance is created, its intra-VPC connection addresses cannot be modified.

For details about accessing DCS instances through clients, see [Accessing a DCS Redis Instance Through redis-cli](#).

7.3.16 Does DCS Support Cross-AZ Deployment?

Master/Standby and cluster DCS Redis instances and DCS Memcached instances can be deployed across availability zones (AZs).

- If instances nodes in an AZ are faulty, nodes in other AZs will not be affected. The standby node automatically becomes the master node to continue to operate, ensuring disaster recovery (DR).
- Cross-AZ deployment does not compromise the speed of data synchronization between the master and standby nodes.

7.3.17 Why Does It Take a Long Time to Start a Cluster DCS Instance?

Possible cause: When a cluster instance is started, status and data are synchronized between the nodes of the instance. If a large amount of data is continuously written into the instance before the synchronization is complete, the synchronization will be prolonged and the instance remains in the **Starting** state. After the synchronization is complete, the instance enters the **Running** state.

Solution: Start writing data to an instance only after the instance has been started.

7.3.18 Does DCS for Redis Provide Backend Management Software?

No. If you wish to query Redis configurations and usage information, use `redis-cli`. If you wish to monitor DCS Redis instance metrics, go to the Cloud Eye console. For details on how to configure and view the metrics, see [Monitoring](#).

7.3.19 Why Is Memory of a DCS Redis Instance Used Up by Just a Few Keys?

Possible cause: The output buffer may have occupied an excessive amount of memory.

Solution: After connecting to the instance using `redis-cli`, run the `redis-cli --bigkeys` command to scan for big keys. Then, run the `info` command to check the output buffer size.

7.3.20 Can I Recover Data from Deleted DCS Instances?

If a DCS instance is automatically deleted or manually deleted through the Redis client, its data cannot be retrieved. If you have backed up the instance, you can

restore its data from the backup. However, the restoration will overwrite the data written in during the period from the backup and the restoration.

By default, data is not evicted from DCS instances. You can modify the instance configuration parameters to adjust the eviction policy so that the instance can evict key values.

7.3.21 Why Is "Error in execution" Returned When I Access Redis?

Symptom: "Error in execution; nested exception is io.lettuce.core.RedisCommandExecutionException: OOM command not allowed when used memory > 'maxmemory'" is returned during a Redis connection.

Analysis: An out-of-memory (OOM) error indicates that the maximum memory is exceeded. In the error information, the "maxmemory" parameter indicates the maximum memory configured on the Redis server.

If the memory usage of the Redis instance is less than 100%, the memory of the node where data is written may have reached the maximum limit. Connect to each node in the cluster by running **redis-cli -h <redis_ip> -p 6379 -a <redis_password> -c --bigkeys**. When connecting to a replica node, run the **READONLY** command before running the **bigkeys** command.

7.4 Redis Commands

7.4.1 How Do I Clear Redis Data?

Exercise caution when clearing data.

- Redis 3.0

Data of a DCS Redis 3.0 instance cannot be cleared on the console, and can only be cleared by the **FLUSHDB** or **FLUSHALL** command in redis-cli.

Run the **FLUSHALL** command to clear all the data in the instance.

Run the **FLUSHDB** command to clear the data in the currently selected DB.

- Redis 4.0 and 5.0

To clear data of a DCS Redis 4.0 or 5.0 instance, you can run the **FLUSHDB** or **FLUSHALL** command in redis-cli, use the data clearing function on the DCS console, or run the **FLUSHDB** command on Web CLI.

To clear data of a Redis Cluster instance, run the **FLUSHDB** or **FLUSHALL** command on every shard of the instance. Otherwise, data may not be completely cleared.

 **NOTE**

- Currently, only DCS Redis 4.0 and 5.0 instances support data clearing by using the DCS console and by running the **FLUSHDB** command on Web CLI.
- When you run the **FLUSHDB** command on Web CLI, only one shard is cleared at a time. If there are multiple shards, connect to and run the **FLUSHDB** command on each master node.
- Redis Cluster data cannot be cleared by using Web CLI.

7.4.2 How Do I Rename High-Risk Commands?

Currently, you can only rename critical commands **COMMAND**, **KEYS**, **FLUSHDB**, **FLUSHALL**, and **HGETALL** for DCS Redis 4.0 and 5.0 instances.

Rename them during instance creation or on the console after the instance is created. To do so, choose **More > Command Renaming** in the instance list.

7.4.3 Does DCS for Redis Support Pipelining?

Cross-region image pull over public network is supported. For DCS Redis 4.0 and 5.0 instances in the Redis Cluster mode, ensure that all commands in a pipeline are executed on the same shard.

7.4.4 Does DCS for Redis Support the INCR and EXPIRE Commands?

Yes. For more information about Redis command compatibility, see *Distributed Cache Service (DCS) 1.9.0 Service Overview (for Huawei Cloud Stack 8.2.0)*.

7.4.5 Why Do I Fail to Execute Some Redis Commands?

Possible causes include the following:

- The command is incorrect.
- The command is disabled in DCS.
For security purposes, some Redis commands are disabled in DCS. For details about disabled and restricted Redis commands, see [Command Compatibility](#).
- The LUA script fails to be executed.
For example, the error message "ERR unknown command 'EVAL'" indicates that your DCS Redis instance is of a lower version that does not support the LUA script. In this case, contact technical support for the instance to be upgraded.
- The **CLIENT SETNAME** and **CLIENT GETNAME** commands fail to be executed.
This is because the DCS Redis instance is of a lower version that does not support these commands. In this case, contact technical support for the instance to be upgraded.

7.4.6 Why Does a Redis Command Fail to Take Effect?

Run the command in redis-cli to check whether the command takes effect.

The following describes two scenarios:

- Scenario 1: Set and query the value of a key to check whether the **SET** and **GET** commands work.
The **SET** command is used to set the string value. If the value is not changed, run the following commands in redis-cli to access the instance:

```
192.168.2.2:6379> set key_name key_value
OK
192.168.2.2:6379> get key_name
"key_value"
192.168.2.2:6379>
```

- Scenario 2: If the timeout set using the **EXPIRE** command is incorrect, perform the following operations:

Set the timeout to 10 seconds and run the **TTL** command to view the remaining time. As shown in the following example, the remaining time is 7 seconds.

```
192.168.2.2:6379> expire key_name 10
(integer) 1
192.168.2.2:6379> ttl key_name
(integer) 7
192.168.2.2:6379>
```

NOTE

Redis clients (including redis-cli, Jedis clients, and Python clients) communicate with Redis server using a binary protocol.

If Redis commands are run properly in redis-cli, the problem may lie in the service code. In this case, create logs in the code for further analysis.

7.4.7 Is There a Time Limit on Executing Redis Commands? What Will Happen If a Command Times Out?

The time limit for executing a Redis command is 1 minute. This limit cannot be configured. After the execution of a command times out, your client will be automatically disconnected.

7.5 Instance Scaling and Upgrade

7.5.1 Can DCS Redis Instances Be Upgraded, for Example, from Redis 3.0 to Redis 4.0 or 5.0?

No. Different Redis versions use different underlying architectures. The Redis version used by a DCS instance cannot be changed once the instance is created. However, you will be informed of any defects or problems found in Redis.

If your service requires the features of higher Redis versions, create a DCS Redis instance of a higher version and then migrate data from the original instance to the new one. For details on how to migrate data, see [Migrating Data with DCS](#).

7.5.2 Are Services Interrupted If Maintenance is Performed During the Maintenance Time Window?

O&M personnel will contact you before performing maintenance during the maintenance time window, informing you of the operations and their impacts. You do not need to worry about instance running exceptions.

7.5.3 Are Instance Resources Affected During Specification Modification?

No. Specification modifications can take place while the instance is running and do not affect any other resources.

7.5.4 Are Services Interrupted During Specification Modification?

You are advised to change the instance specifications during off-peak hours because specification modification has the following impacts:

- **Impact of instance type changes:**
 - From single-node to master/standby for a DCS Redis 3.0 instance:
The instance cannot be connected for several seconds and remains read-only for about 1 minute.
 - From master/standby to Proxy Cluster for a DCS Redis 3.0 instance:
The instance cannot be connected and remains read-only for 5 to 30 minutes.
- **Impact of capacity expansion and reduction:**
 - Single-node and master/standby
The DCS Redis 3.0, 4.0, or 5.0 instance cannot be connected for several seconds and remains read-only for about 1 minute.
For capacity expansion, only the memory of the instance is expanded. The CPU processing capability is not improved.
Data of single-node instances may not be retained because they do not support data persistence. After the scaling, check whether the data is complete and import data if required.
 - Proxy Cluster
The instance can be connected, but the CPU will be occupied and the latency will increase during data migration. During capacity expansion, new Redis Server nodes are added, and data is automatically balanced to the new nodes.
 - Redis Cluster
The instance can be connected, but the CPU usage and latency will increase during data migration. During capacity expansion, new Redis Server nodes are added, and data is automatically balanced to the new nodes.
 - Backup records created before the capacity change cannot be restored.

7.5.5 Why Can't I Modify Specifications for a DCS Redis/Memcached Instance?

Specifications of a DCS instance cannot be modified if another task of the instance is still running. For example, you cannot delete or scale up an instance while it is being restarted. Likewise, you cannot delete an instance while it is being scaled up.

If the specification modification fails, try again later. If it fails again, contact technical support.

7.6 Monitoring and Alarm

7.6.1 Does Redis Support Command Audits?

No. To ensure high-performance reads and writes, Redis does not audit commands. Commands are not printed.

7.6.2 What Should I Do If the Monitoring Data of a DCS Redis Instance Is Abnormal?

If you have any doubt on the monitoring data of a DCS Redis instance, you can access the instance through redis-cli and run the **INFO ALL** command to view the metrics. For details about the output of the **INFO ALL** command, see <http://www.redis.io/commands/info>.

7.6.3 Why Is Available Memory of Unused DCS Instances Less Than Total Memory and Why Is Memory Usage of Unused DCS Instances Greater Than Zero?

The available memory is less than the total memory because some memory is reserved for system overhead and data persistence (supported by master/standby instances). DCS instances use a certain amount of memory for Redis-server buffers and internal data structures. This is why memory usage of unused DCS instances is greater than zero.

7.6.4 Why Is Used Memory Greater Than Available Memory?

For single-node and master/standby DCS instances, the used instance memory is measured by the Redis-server process. For cluster DCS instances, the used cluster memory is the sum of used memory of all shards in the cluster.

Due to the internal implementation of the open-source redis-server, the used instance memory is normally slightly higher than the available instance memory.

Why is used_memory higher than max_memory?

Redis allocates memory using zmalloc. It does not check whether used_memory exceeds max_memory every time the memory is allocated. Instead, it checks whether the current used_memory exceeds max_memory at the beginning of a periodic task or command processing. If used_memory exceeds max_memory, eviction is triggered. Therefore, the restrictions of the max_memory policy are not implemented in real time or rigidly. A case in which the used_memory is greater than the max_memory may occur occasionally.

7.7 Data Backup, Export, and Migration

7.7.1 How Do I Export DCS Redis Instance Data?

- For master/standby or cluster instances:
 - Perform the following operations to export the data:
 - a. On the **Backups & Restorations** page, view the backup records.

- b. If there are no backup records, create a backup manually and download the backup file as prompted.

 **NOTE**

If your DCS instances were created a long time ago, the versions of these instances may not be advanced enough to support some new functions (such as backup and restoration). You can contact technical support to upgrade your DCS instances. After the upgrade, you can back up and restore your instances.

- For single-node instances:

Single-node instances do not support the backup function. You can use `redis-cli` to export RDB files. This operation depends on **SYNC** command.

- If the instance allows the **SYNC** command (such as a Redis 3.0 single-node instance), run the following command to export the instance data:

```
redis-cli -h {source_redis_address} -p 6379 [-a password] --rdb {output.rdb}
```

- If the instance does not allow the **SYNC** command (such as a Redis 4.0 or 5.0 single-node instance), migrate the instance data to a master/standby instance and export the data by using the backup function.

7.7.2 Can I Export Backup Data of DCS Redis Instances to RDB Files Using the Console?

- Redis 3.0

No. On the console, backup data of a DCS Redis 3.0 instance can be exported only to AOF files. To export data to RDB files, run the following command in `redis-cli`:

```
redis-cli -h {redis_address} -p 6379 [-a password] --rdb {output.rdb}
```

- Redis 4.0 and 5.0

Yes. Backup data of a DCS Redis 4.0 or 5.0 instance is exported from the console to RDB files. You cannot use `redis-cli` to export such data to RDB files.

7.7.3 Does DCS Support Data Persistence?

DCS Redis instances:

- Single-node: Not supported
- Master/Standby and cluster: Supported

DCS Memcached instances:

- Single-node: Not supported
- Master/Standby: Supported

7.7.4 Online Migration with Rump

Background

Rump is an open-source tool designed for migrating Redis data online. It supports migration between DBs of the same instance and between DBs of different instances.

Migration Principles

Rump uses the **SCAN** command to acquire keys and the **DUMP/RESTORE** command to get or set values.

Featuring time complexity $O(1)$, **SCAN** is capable of quickly getting all keys. **DUMP/RESTORE** is used to read/write values independent from the key type.

Rump brings the following benefits:

- The **SCAN** command replaces the **KEYS** command to avoid blocking Redis.
- Any type of data can be migrated.
- **SCAN** and **DUMP/RESTORE** operations are pipelined, improving the network efficiency during data migration.
- No temporary file is involved, saving disk space.
- Buffered channels are used to optimize performance of the source server.

NOTICE

1. To cluster DCS instances, you cannot use Rump. Instead, use `redis-port` or `redis-cli`.
 2. To prevent migration command resolution errors, do not include special characters (`#@:`) in the instance password.
 3. Stop the service before migrating data. If data is kept being written in during the migration, some keys might be lost.
-

Step 1: Installing Rump

1. Download [Rump \(release version\)](#).
On 64-bit Linux, run the following command:

```
wget https://github.com/stickermule/rump/releases/download/0.0.3/rump-0.0.3-linux-amd64;
```
2. After decompression, run the following commands to add the execution permission:

```
mv rump-0.0.3-linux-amd64 rump;  
chmod +x rump;
```

Step 2: Migrating Data

```
rump -from {source_redis_address} -to {target_redis_address}
```

Parameter/Option description:

- `{source_redis_address}`
Source Redis instance address, in the format of `redis://[user:password@]host:port/db`. `[user:password@]` is optional. If the instance is accessed in password-protected mode, you must specify the password in the RFC 3986 format. `user` can be omitted, but the colon (`:`) cannot be omitted. For example, the address may be `redis://:mypassword@192.168.0.45:6379/1`.

db is the sequence number of the database. If it is not specified, the default value is 0.

- `{target_redis_address}`

Address of the target Redis instance, in the same format as the source.

In the following example, data in DBO of the source Redis is migrated to the target Redis whose connection address is 192.168.0.153. ********* stands for the password.

```
[root@ecs ~]# ./rump -from redis://127.0.0.1:6379/0 -to redis://*****@192.168.0.153:6379/0
.Sync done.
[root@ecs ~]#
```

7.8 Master/Standby Switchover

7.8.1 When Does a Master/Standby Switchover Occur?

A master/standby switchover may occur in the following scenarios:

- A master/standby switchover operation is initiated on the DCS Console.
- If the master node of a master/standby instance fails, a master/standby switchover will be triggered.

For example, running commands that consume a lot of resources, such as **KEYS** commands, will cause CPU usage to spike and as result triggers a master/standby switchover.

- If you restart a master/standby instance on the DCS console, a master/standby switchover will be triggered.

After a master/standby switchover occurs, you will receive a notification. Check whether the client services are running properly. If not, check whether the TCP connection is normal and whether it can be re-established after the master/standby switchover to restore the services.

7.8.2 How Does Master/Standby Switchover Affect Services?

If a fault occurs in a master/standby or cluster DCS instance, a failover is triggered automatically. Services may be interrupted for less than half a minute during exception detection and failover.

7.8.3 Does the Client Need to Switch the Connection Address After a Master/Standby Switchover?

No. If the master fails, the standby node will be promoted to master and takes the original IP address.

7.8.4 How Does Redis Master/Standby Replication Work?

Redis master/standby instances are also called master/slave instances. Generally, updates to the master cache node are automatically and asynchronously replicated to the standby cache node. This means that data in the standby cache node may not always be consistent with data in the master cache node. The inconsistency is typically seen when the I/O write speed of the master node is

faster than the synchronization speed of the standby node or a network latency occurs between the master and standby nodes. If a failover happens when some data is not yet replicated to the standby node, such data may be lost after the failover.

7.9 Memcached Usage

7.9.1 Can I Dump DCS Memcached Instance Data for Analysis?

No.

7.9.2 What Memcached Version Is Compatible with DCS for Memcached?

DCS for Memcached is based on Redis 3.0.7 and is compatible with Memcached 1.5.1.

7.9.3 What Data Structures Does DCS for Memcached Support?

Only the key-value structure is supported.

7.9.4 Does DCS for Memcached Support Public Access?

No. The ECS that serves as a client and the DCS instance that the client will access must belong to the same VPC. In the application development and debugging phase, you can also use an SSH agent to access DCS instances in the local environment.

7.9.5 Can I Modify Configuration Parameters of DCS Memcached Instances?

Parameter configuration is allowed only when DCS instances are in the **Running** state.

For details, see [Modifying Configuration Parameters](#).

7.9.6 What Are the Differences Between DCS for Memcached and Self-Hosted Memcached?

[Table 7-5](#) describes the differences between DCS for Memcached and self-hosted Memcached.

Table 7-5 Comparing DCS for Memcached and self-hosted Memcached

Item	DCS Memcached	Self-Hosted Memcached
Confirming Deployment	Easy to deploy. DCS for Memcached can be used right out of the box without requiring you to worry about the hardware or software.	Involves complicated operations and settings.
Availability	Master/Standby instances use hot standby to ensure stable services. If the master node is faulty, the standby cache node will automatically become the master node to prevent a single point of failure.	Requires additional configurations.
Security	Uses the VPC and security groups for network access security control.	Requires you to design and implement a security mechanism by yourself.
Scale-up	Supports online scale-up on the console.	Requires additional hardware and restarting your service.

7.9.7 What Policies Does DCS for Memcached Use to Deal with Expired Data?

DCS for Memcached allows you to set the expiration time for stored data based on service requirements. For example, you can set the **expire** time when performing the **add** operation.

```

> help add
Synopsis: add <key> <value> <expire>

Options:
  • <key> (string, required)
    add key
  • <value> (string, required)
    add value
  • <expire> (string, required)
  
```

By default, data is not evicted from DCS Memcached instances. In the current version of DCS for Memcached, you can select an eviction policy.

For details about the six types of data eviction policies, see [What Is the Default Data Eviction Policy?](#)

7.9.8 How Should I Select AZs When Creating a DCS Memcached Instance?

Different AZs within a region do not differ in functions.

Generally, instance deployment within an AZ features lower network latency while cross-AZ deployment ensures disaster recovery. If your application requires lower network latency, choose single-AZ deployment.

DCS for Memcached supports cross-AZ deployment. When creating a DCS Memcached instance on the DCS console, you can select any AZ in the same region as your ECS for communication between your ECS and instance. For lower network latency, select the same AZ as your ECS.

Note that there may be only one available AZ due to insufficient resources when you create a DCS Memcached instance. This does not affect the normal use of DCS.

A Change History

Table A-1 Change history

Released On	Description
2022-04-12	This issue is the first official release.